**1. Why UDP over TCP?**

**Finer application-level control over what data is sent, and when.** Under UDP, as soon as an application process passes data to UDP, UDP will package the data inside a UDP segment and immediately pass the segment to the network layer. TCP, on the other hand, has a congestion control mechanism that throttles the transport-layer TCP sender when one or more links between the source and destination hosts become excessively congested. TCP will also continue to resend a segment until the receipt of the segment has been acknowledged by the destination, regardless of how long reliable delivery takes. Since real-time applications often require a minimum sending rate, do not want to overly delay segment transmission, and can tolerate some data loss, TCP’s service model is not particularly well matched to these applications’ needs. As discussed below, these applications can use UDP and implement, as part of the application, any additional functionality that is needed beyond UDP’s no-frills segment-delivery service.

**No connection establishment.** As we’ll discuss later, TCP uses a three-way handshake before it starts to transfer data. UDP just blasts away without any formal preliminaries. Thus UDP does not introduce any delay to establish a connection. This is probably the principal reason why DNS runs over UDP rather than TCP—DNS would be much slower if it ran over TCP. HTTP uses TCP rather than UDP, since reliability is critical for Web pages with text. But, as we briefly discussed in Section 2.2, the TCP connection-establishment delay in HTTP is an important contributor to the delays associated with downloading Web documents. Indeed, the QUIC protocol (Quick UDP Internet Connection, [Iyengar 2015]), used in Google’s Chrome browser, uses UDP as its underlying transport protocol and implements reliability in an application-layer protocol on top of UDP.

**No connection state.** TCP maintains connection state in the end systems. This connection state includes receive and send buffers, congestion-control parameters, and sequence and acknowledgment number parameters. We will see in Section 3.5 that this state information is needed to implement TCP’s reliable data transfer service and to provide congestion control. UDP, on the other hand, does not maintain connection state and does not track any of these parameters. For this reason, a server devoted to a particular application can typically support many more active clients when the application runs over UDP rather than TCP.

**Small packet header overhead.** The TCP segment has 20 bytes of header overhead in every segment, whereas UDP has only 8 bytes of overhead.

**2. Why does UDP have checksum?**

You may wonder why UDP provides a checksum in the first place, as many link-layer protocols (including the popular Ethernet protocol) also provide error checking. The reason is that there is no guarantee that all the links between source and destination provide error checking; that is, one of the links may use a link-layer protocol that does not provide error checking. Furthermore, even if segments are correctly transferred across a link, it’s possible that bit errors could be introduced when a segment is stored in a router’s memory. Given that neither link-by-link reliability nor in-memory error detection is guaranteed, UDP must provide error detection at the transport layer, on an end-end basis, if the endend data transfer service is to provide error detection. This is an example of the celebrated **end-end principle** in system design.

**3. Difference between connection set up in transport layer and connection set up (virtual circuits) in the network layer**.

There is a subtle but important distinction between VC setup at the network layer and connection setup at the transport layer (for example, the TCP three-way handshake we studied in Chapter 3). Connection setup at the transport layer involves only the two end systems. During transport-layer connection setup, the two end systems alone determine the parameters (for example, initial sequence number and flow-control window size) of their transport-layer connection. Although the two end systems are aware of the transport-layer connection, the routers within the network are completely oblivious to it. On the other hand, with a VC network layer, routers along the path between the two end systems are involved in VC setup, and each router is fully aware of all the VCs passing through it.

**4. Some of the advances embodied in OSPF include the following**:

**Security**. Exchanges between OSPF routers (for example, link-state updates) can be authenticated. With authentication, only trusted routers can participate in the OSPF protocol within an AS, thus preventing malicious intruders (or networking students taking their newfound knowledge out for a joyride) from injecting incorrect information into router tables. By default, OSPF packets between routers are not authenticated and could be forged. Two types of authentication can be configured— simple and MD5 (see Chapter 8 for a discussion on MD5 and authentication in general). With simple authentication, the same password is configured on each router. When a router sends an OSPF packet, it includes the password in plaintext. Clearly, simple authentication is not very secure. MD5 authentication is based on shared secret keys that are configured in all the routers. For each OSPF packet that it sends, the router computes the MD5 hash of the content of the OSPF packet appended with the secret key. (See the discussion of message authentication codes in Chapter 8.) Then the router includes the resulting hash value in the OSPF packet. The receiving router, using the preconfigured secret key, will compute an MD5 hash of the packet and compare it with the hash value that the packet carries, thus verifying the packet’s authenticity. Sequence numbers are also used with MD5 authentication to protect against replay attacks.

**Multiple same-cost paths.** When multiple paths to a destination have the same cost, OSPF allows multiple paths to be used (that is, a single path need not be chosen for carrying all traffic when multiple equal-cost paths exist).

**Integrated support for unicast and multicast routing.** Multicast OSPF (MOSPF) [RFC 1584] provides simple extensions to OSPF to provide for multicast routing. MOSPF uses the existing OSPF link database and adds a new type of link-state advertisement to the existing OSPF link-state broadcast mechanism.

**Support for hierarchy within a single AS.** An OSPF autonomous system can be configured hierarchically into areas. Each area runs its own OSPF link-state routing algorithm, with each router in an area broadcasting its link state to all other routers in that area. Within each area, one or more area border routers are responsible for routing packets outside the area. Lastly, exactly one OSPF area in the AS is configured to be the backbone area. The primary role of the backbone area is to route traffic between the other areas in the AS. The backbone always contains all area border routers in the AS and may contain non-border routers as well. Inter-area routing within the AS requires that the packet be first routed to an area border router (intra-area routing), then routed through the backbone to the area border router that is in the destination area, and then routed to the final destination.

**5. As an inter-AS routing protocol, BGP provides each router a means to:**

**1. Obtain prefix reachability information from neighboring ASs.** In particular, BGP allows each subnet to advertise its existence to the rest of the Internet. A subnet screams, “I exist and I am here,” and BGP makes sure that all the routers in the Internet know about this subnet. If it weren’t for BGP, each subnet would be an isolated island—alone, unknown and unreachable by the rest of the Internet.

**2. Determine the “best” routes to the prefixes**. A router may learn about two or more different routes to a specific prefix. To determine the best route, the router will locally run a BGP routeselection procedure (using the prefix reachability information it obtained via neighboring routers). The best route will be determined based on policy as well as the reachability information.

**6. TCP features:**

TCP, on the other hand, offers several additional services to applications. First and foremost, it provides reliable data transfer. Using flow control, sequence numbers, acknowledgments, and timers (techniques we’ll explore in detail in this chapter), TCP ensures that data is delivered from sending process to receiving process, correctly and in order. TCP thus converts IP’s unreliable service between end systems into a reliable data transport service between processes. TCP also provides congestion control. Congestion control is not so much a service provided to the invoking application as it is a service for the Internet as a whole, a service for the general good. Loosely speaking, TCP congestion control prevents any one TCP connection from swamping the links and routers between communicating hosts with an excessive amount of traffic. TCP strives to give each connection traversing a congested link an equal share of the link bandwidth. This is done by regulating the rate at which the sending sides of TCP connections can send traffic into the network. UDP traffic, on the other hand, is unregulated. An application using UDP transport can send at any rate it pleases, for as long as it pleases.

**7.** Suppose the network layer provides the following service. The network layer in the source host accepts a segment of maximum size 1,200 bytes and a destination host address from the transport layer. The network layer then guarantees to deliver the segment to the transport layer at the destination host. Suppose many network application processes can be running at the destination host. a. Design the simplest possible transport-layer protocol that will get application data to the desired process at the destination host. Assume the operating system in the destination host has assigned a 4-byte port number to each running application process. b. Modify this protocol so that it provides a “return address” to the destination process. c. In your protocols, does the transport layer “have to do anything” in the core of the computer network?

a) Call this protocol Simple Transport Protocol (STP). At the sender side, STP accepts from the sending process a chunk of data not exceeding 1196 bytes, a destination host address, and a destination port number. STP adds a four-byte header to each chunk and puts the port number of the destination process in this header. STP then gives the destination host address and the resulting segment to the network layer. The network layer delivers the segment to STP at the destination host. STP then examines the port number in the segment, extracts the data from the segment, and passes the data to the process identified by the port number.

b) The segment now has two header fields: a source port field and destination port field. At the sender side, STP accepts a chunk of data not exceeding 1192 bytes, a destination host address, a source port number, and a destination port number. STP creates a segment which contains the application data, source port number, and destination port number. It then gives the segment and the destination host address to the network layer. After receiving the segment, STP at the receiving host gives the application process the application data and the source port number.

c) No, the transport layer does not have to do anything in the core; the transport layer “lives” in the end systems.

**8.** In protocol rdt3.0 , the ACK packets flowing from the receiver to the sender do not have sequence numbers (although they do have an ACK field that contains the sequence number of the packet they are acknowledging). Why is it that our ACK packets do not require sequence numbers?

To best answer this question, consider why we needed sequence numbers in the first place. We saw that the sender needs sequence numbers so that the receiver can tell if a data packet is a duplicate of an already received data packet. In the case of ACKs, the sender does not need this info (i.e., a sequence number on an ACK) to tell detect a duplicate ACK. A duplicate ACK is obvious to the rdt3.0 receiver, since when it has received the original ACK it transitioned to the next state. The duplicate ACK is not the ACK that the sender needs and hence is ignored by the rdt3.0 sender.

**9.** Consider a channel that can lose packets but has a maximum delay that is known. Modify protocol rdt2.1 to include sender timeout and retransmit. Informally argue why your protocol can communicate correctly over this channel.

Here, we add a timer, whose value is greater than the known round-trip propagation delay. We add a timeout event to the “Wait for ACK or NAK0” and “Wait for ACK or NAK1” states. If the timeout event occurs, the most recently transmitted packet is retransmitted. Let us see why this protocol will still work with the rdt2.1 receiver.

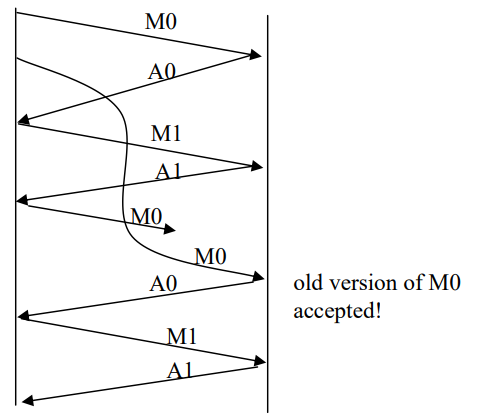
• Suppose the timeout is caused by a lost data packet, i.e., a packet on the senderto-receiver channel. In this case, the receiver never received the previous transmission and, from the receiver's viewpoint, if the timeout retransmission is received, it looks exactly the same as if the original transmission is being received.

• Suppose now that an ACK is lost. The receiver will eventually retransmit the packet on a timeout. But a retransmission is exactly the same action that if an ACK is garbled. Thus the sender's reaction is the same with a loss, as with a garbled ACK. The rdt 2.1 receiver can already handle the case of a garbled ACK.

**10.** The sender side of rdt3.0 simply ignores (that is, takes no action on) all received packets that are either in error or have the wrong value in the acknum field of an acknowledgment packet. Suppose that in such circumstances, rdt3.0 were simply to retransmit the current data packet. Would the protocol still work?

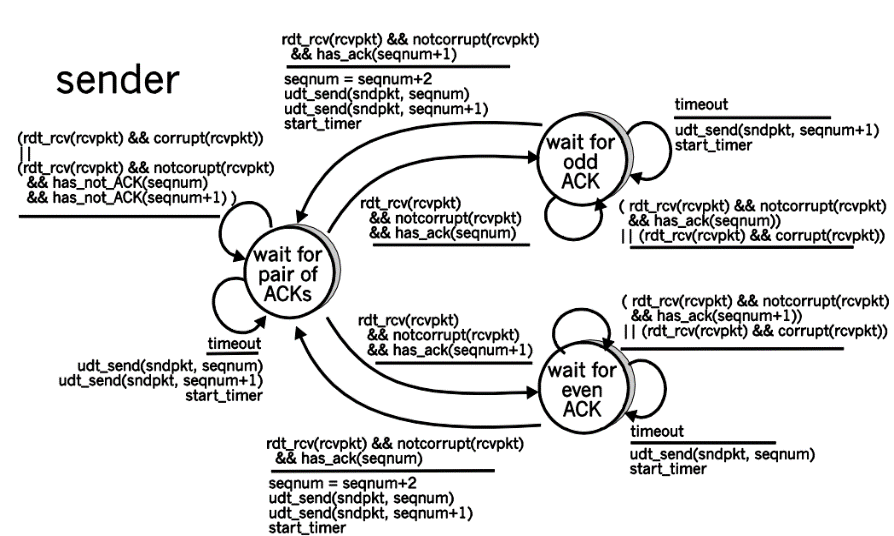
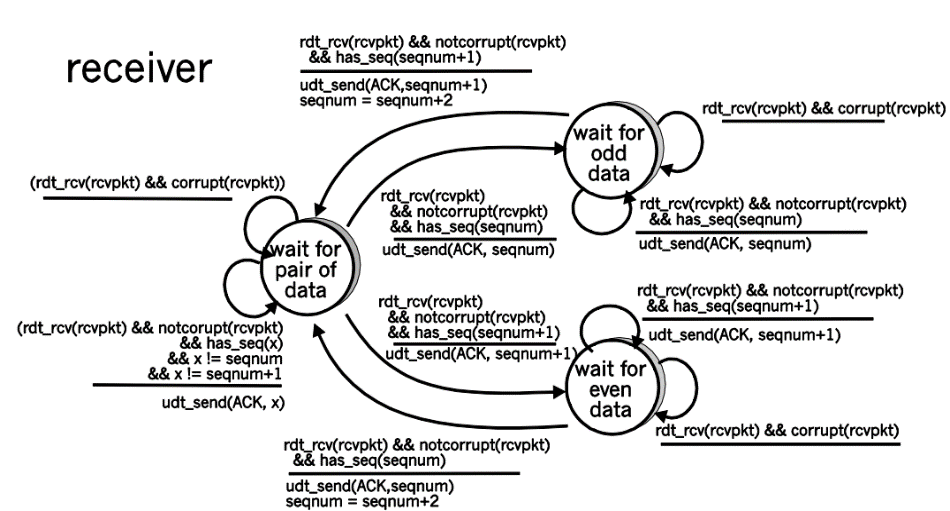
The protocol would still work, since a retransmission would be what would happen if the packet received with errors has actually been lost (and from the receiver standpoint, it never knows which of these events, if either, will occur). To get at the more subtle issue behind this question, one has to allow for premature timeouts to occur. In this case, if each extra copy of the packet is ACKed and each received extra ACK causes another extra copy of the current packet to be sent, the number of times packet n is sent will increase without bound as n approaches infinity.

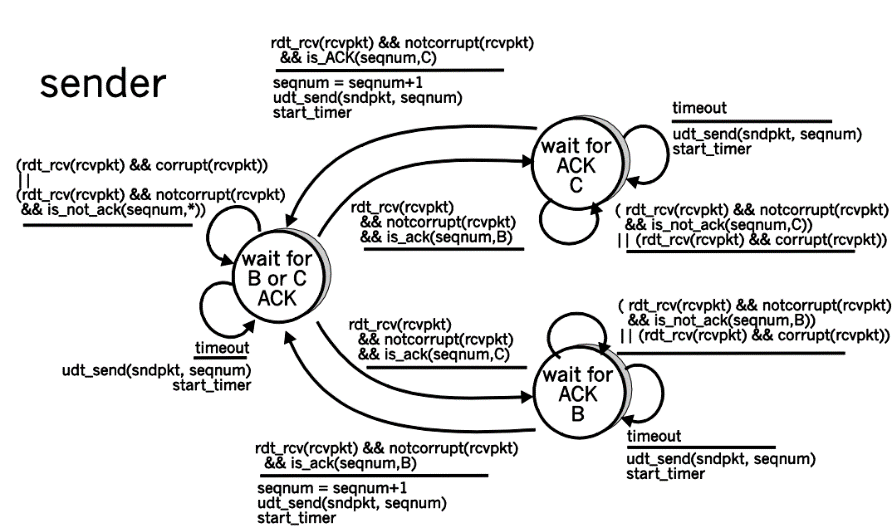
**11.** Consider the rdt 3.0 protocol. Draw a diagram showing that if the network connection between the sender and receiver can reorder messages (that is, that two messages propagating in the medium between the sender and receiver can be reordered), then the alternating-bit protocol will not work correctly (make sure you clearly identify the sense in which it will not work correctly).

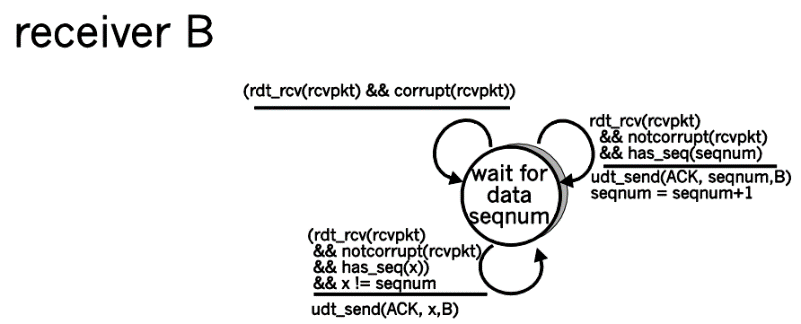


**12.** Suppose an application uses rdt 3.0 as its transport layer protocol. As the stop-and-wait protocol has very low channel utilization (shown in the cross-country example), the designers of this application let the receiver keep sending back a number (more than two) of alternating ACK 0 and ACK 1 even if the corresponding data have not arrived at the receiver. Would this application design increase the channel utilization? Why? Are there any potential problems with this approach? Explain.

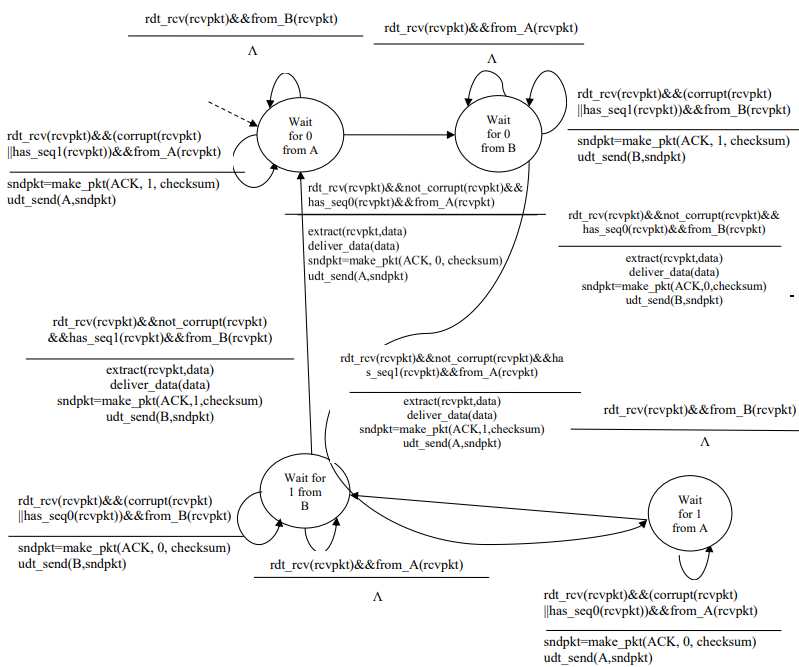
Yes. This actually causes the sender to send a number of pipelined data into the channel. Yes. Here is one potential problem. If data segments are lost in the channel, then the sender of rdt 3.0 won’t re-send those segments, unless there are some additional mechanism in the application to recover from loss.

**13.** In the generic SR protocol that we studied in Section 3.4.4 , the sender transmits a message as soon as it is available (if it is in the window) without waiting for an acknowledgment. Suppose now that we want an SR protocol that sends messages two at a time. That is, the sender will send a pair of messages and will send the next pair of messages only when it knows that both messages in the first pair have been received correctly. Suppose that the channel may lose messages but will not corrupt or reorder messages. Design an error-control protocol for the unidirectional reliable transfer of messages.

**14.** Consider a scenario in which Host A wants to simultaneously send packets to Hosts B and C. A is connected to B and C via a broadcast channel—a packet sent by A is carried by the channel to both B and C. Suppose that the broadcast channel connecting A, B, and C can independently lose and corrupt packets (and so, for example, a packet sent from A might be correctly received by B, but not by C). Design a stop-and-wait-like error-control protocol for reliably transferring packets from A to B and C, such that A will not get new data from the upper layer until it knows that both B and C have correctly received the current packet.

****

**15.** Consider a scenario in which Host A and Host B want to send messages to Host C. Hosts A and C are connected by a channel that can lose and corrupt (but not reorder) messages. Hosts B and C are connected by another channel (independent of the channel connecting A and C) with the same properties. The transport layer at Host C should alternate in delivering messages from A and B to the layer above (that is, it should first deliver the data from a packet from A, then the data from a packet from B, and so on). Design a stop-and-wait-like error-control protocol for reliably transferring packets from A and B to C, with alternating delivery at C.



**16.** Suppose we have two network entities, A and B. B has a supply of data messages that will be sent to A according to the following conventions. When A gets a request from the layer above to get the next data (D) message from B, A must send a request (R) message to B on the A-to-B channel. Only when B receives an R message can it send a data (D) message back to A on the B-to-A channel. A should deliver exactly one copy of each D message to the layer above. R messages can be lost (but not corrupted) in the A-to-B channel; D messages, once sent, are always delivered correctly. The delay along both channels is unknown and variable.

Because the A-to-B channel can lose request messages, A will need to timeout and retransmit its request messages (to be able to recover from loss). Because the channel delays are variable and unknown, it is possible that A will send duplicate requests (i.e., resend a request message that has already been received by B). To be able to detect duplicate request messages, the protocol will use sequence numbers. A 1-bit sequence number will suffice for a stop-and-wait type of request/response protocol. A (the requestor) has 4 states:

• “Wait for Request 0 from above.” Here the requestor is waiting for a call from above to request a unit of data. When it receives a request from above, it sends a request message, R0, to B, starts a timer and makes a transition to the “Wait for D0” state. When in the “Wait for Request 0 from above” state, A ignores anything it receives from B.

• “Wait for D0”. Here the requestor is waiting for a D0 data message from B. A timer is always running in this state. If the timer expires, A sends another R0 message, restarts the timer and remains in this state. If a D0 message is received from B, A stops the time and transits to the “Wait for Request 1 from above” state. If A receives a D1 data message while in this state, it is ignored.

• “Wait for Request 1 from above.” Here the requestor is again waiting for a call from above to request a unit of data. When it receives a request from above, it sends a request message, R1, to B, starts a timer and makes a transition to the “Wait for D1” state. When in the “Wait for Request 1 from above” state, A ignores anything it receives from B.

• “Wait for D1”. Here the requestor is waiting for a D1 data message from B. A timer is always running in this state. If the timer expires, A sends another R1 message, restarts the timer and remains in this state. If a D1 message is received from B, A stops the timer and transits to the “Wait for Request 0 from above” state. If A receives a D0 data message while in this state, it is ignored.

The data supplier (B) has only two states:

• “Send D0.” In this state, B continues to respond to received R0 messages by sending D0, and then remaining in this state. If B receives a R1 message, then it knows its D0 message has been received correctly. It thus discards this D0 data (since it has been received at the other side) and then transits to the “Send D1” state, where it will use D1 to send the next requested piece of data.

**17.** Consider the GBN and SR protocols. Suppose the sequence number space is of size k. What is the largest allowable sender window that will avoid the occurrence of problems such as that in Figure 3.27 for each of these protocols?

In order to avoid the scenario of Figure 3.27, we want to avoid having the leading edge of the receiver's window (i.e., the one with the “highest” sequence number) wrap around in the sequence number space and overlap with the trailing edge (the one with the "lowest" sequence number in the sender's window). That is, the sequence number space must be large enough to fit the entire receiver window and the entire sender window without this overlap condition. So - we need to determine how large a range of sequence numbers can be covered at any given time by the receiver and sender windows. Suppose that the lowest-sequence number that the receiver is waiting for is packet m. In this case, it's window is [m,m+w-1] and it has received (and ACKed) packet m-1 and the w-1 packets before that, where w is the size of the window. If none of those w ACKs have been yet received by the sender, then ACK messages with values of [m-w,m-1] may still be propagating back. If no ACKs with these ACK numbers have been received by the sender, then the sender's window would be [m-w,m-1]. Thus, the lower edge of the sender's window is m-w, and the leading edge of the receivers window is m+w-1. In order for the leading edge of the receiver's window to not overlap with the trailing edge of the sender's window, the sequence number space must thus be big enough to accommodate 2w sequence numbers. That is, the sequence number space must be at least twice as large as the window size, k ≥ 2w.

**18.** We have said that an application may choose UDP for a transport protocol because UDP offers finer application control (than TCP) of what data is sent in a segment and when. a. Why does an application have more control of what data is sent in a segment? b. Why does an application have more control on when the segment is sent?

a) Consider sending an application message over a transport protocol. With TCP, the application writes data to the connection send buffer and TCP will grab bytes without necessarily putting a single message in the TCP segment; TCP may put more or less than a single message in a segment. UDP, on the other hand, encapsulates in a segment whatever the application gives it; so that, if the application gives UDP an application message, this message will be the payload of the UDP segment. Thus, with UDP, an application has more control of what data is sent in a segment. b) With TCP, due to flow control and congestion control, there may be significant delay from the time when an application writes data to its send buffer until when the data is given to the network layer. UDP does not have delays due to flow control and congestion control.

**19.**Host A and B are directly connected with a 100 Mbps link. There is one TCP connection between the two hosts, and Host A is sending to Host B an enormous file over this connection. Host A can send its application data into its TCP socket at a rate as high as 120 Mbps but Host B can read out of its TCP receive buffer at a maximum rate of 50 Mbps. Describe the effect of TCP flow control.

Since the link capacity is only 100 Mbps, so host A’s sending rate can be at most 100Mbps. Still, host A sends data into the receive buffer faster than Host B can remove data from the buffer. The receive buffer fills up at a rate of roughly 40Mbps. When the buffer is full, Host B signals to Host A to stop sending data by setting RcvWindow = 0. Host A then stops sending until it receives a TCP segment with RcvWindow > 0. Host A will thus repeatedly stop and start sending as a function of the RcvWindow values it Ack = 247 Ack = 247 Seq = 127, 80 bytes Seq = 127, 80 bytes Seq = 207, 40 bytes Ack = 207 Host A Host B Timeout interval Timeout interval receives from Host B. On average, the long-term rate at which Host A sends data to Host B as part of this connection is no more than 60Mbps.

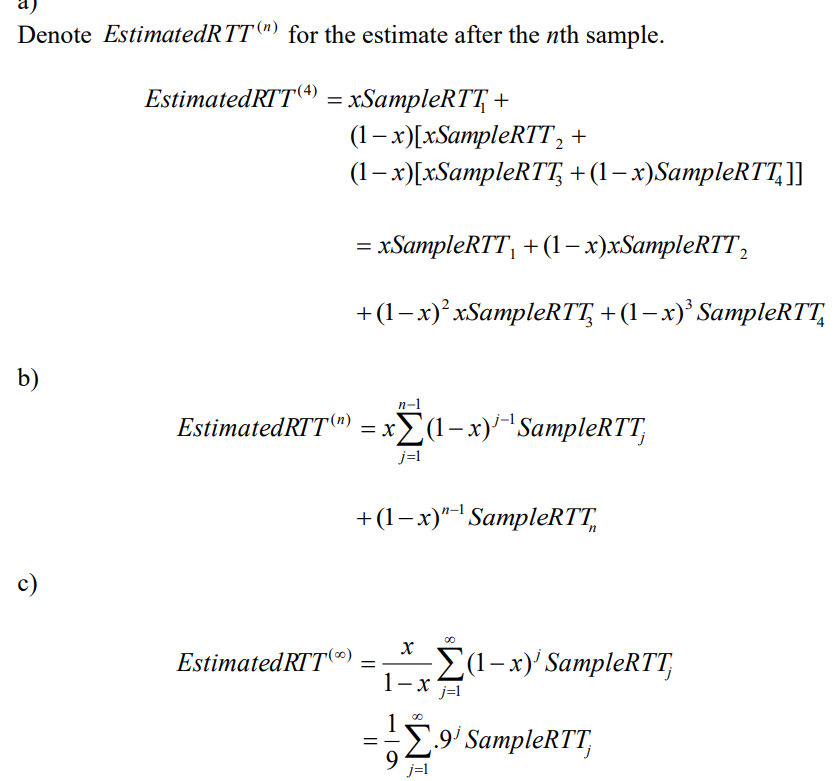
**20.**SYN cookies were discussed in Section 3.5.6 . a. Why is it necessary for the server to use a special initial sequence number in the SYNACK? b. Suppose an attacker knows that a target host uses SYN cookies. Can the attacker create half-open or fully open connections by simply sending an ACK packet to the target? Why or why not? c. Suppose an attacker collects a large amount of initial sequence numbers sent by the server. Can the attacker cause the server to create many fully open connections by sending ACKs with those initial sequence numbers? Why?

a) The server uses special initial sequence number (that is obtained from the hash of source and destination IPs and ports) in order to defend itself against SYN FLOOD attack.

b) No, the attacker cannot create half-open or fully open connections by simply sending and ACK packet to the target. Half-open connections are not possible since a server using SYN cookies does not maintain connection variables and buffers for any connection before full connections are established. For establishing fully open connections, an attacker should know the special initial sequence number corresponding to the (spoofed) source IP address from the attacker. This sequence number requires the "secret" number that each server uses. Since the attacker does not know this secret number, she cannot guess the initial sequence number.

c) No, the sever can simply add in a time stamp in computing those initial sequence numbers and choose a time to live value for those sequence numbers, and discard expired initial sequence numbers even if the attacker replay them.

**21.** Consider the TCP procedure for estimating RTT. Suppose that . Let SampleRTT be the most recent sample RTT, let SampleRTT be the next most recent sample RTT, and so on. a. For a given TCP connection, suppose four acknowledgments have been returned with corresponding sample RTTs: SampleRTT , SampleRTT , SampleRTT , and SampleRTT . Express EstimatedRTT in terms of the four sample RTTs. b. Generalize your formula for n sample RTTs. c. For the formula in part (b) let n approach infinity. Comment on why this averaging procedure is called an exponential moving average.



**22.** Why do you think TCP avoids measuring the SampleRTT for retransmitted segments?

Let’s look at what could wrong if TCP measures SampleRTT for a retransmitted segment. Suppose the source sends packet P1, the timer for P1 expires, and the source then sends P2, a new copy of the same packet. Further suppose the source measures SampleRTT for P2 (the retransmitted packet). Finally suppose that shortly after transmitting P2 an acknowledgment for P1 arrives. The source will mistakenly take this acknowledgment as an acknowledgment for P2 and calculate an incorrect value of SampleRTT.

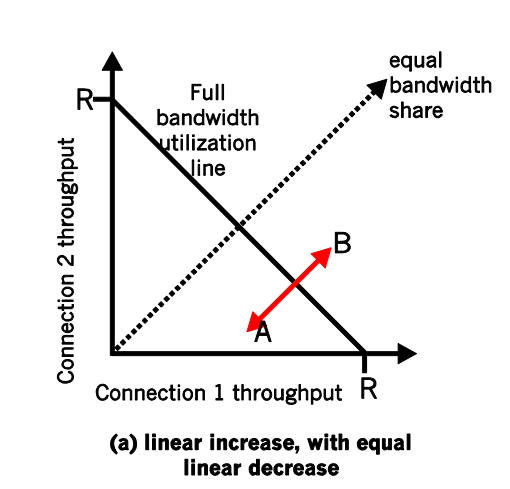
**23.** What is the relationship between the variable SendBase and the variable LastByteRcvd?

At any given time t, SendBase – 1 is the sequence number of the last byte that the sender knows has been received correctly, and in order, at the receiver. The actually last byte received (correctly and in order) at the receiver at time t may be greater if there are acknowledgements in the pipe. Thus SendBase–1 ≤ LastByteRcvd.

**24.** Why do you think the TCP designers chose not to perform a fast retransmit after the first duplicate ACK for a segment is received?

Suppose packets n, n+1, and n+2 are sent, and that packet n is received and ACKed. If packets n+1 and n+2 are reordered along the end-to-end-path (i.e., are received in the order n+2, n+1) then the receipt of packet n+2 will generate a duplicate ack for n and would trigger a retransmission under a policy of waiting only for second duplicate ACK for retransmission. By waiting for a triple duplicate ACK, it must be the case that two packets after packet n are correctly received, while n+1 was not received. The designers of the triple duplicate ACK scheme probably felt that waiting for two packets (rather than 1) was the right tradeoff between triggering a quick retransmission when needed, but not retransmitting prematurely in the face of packet reordering.

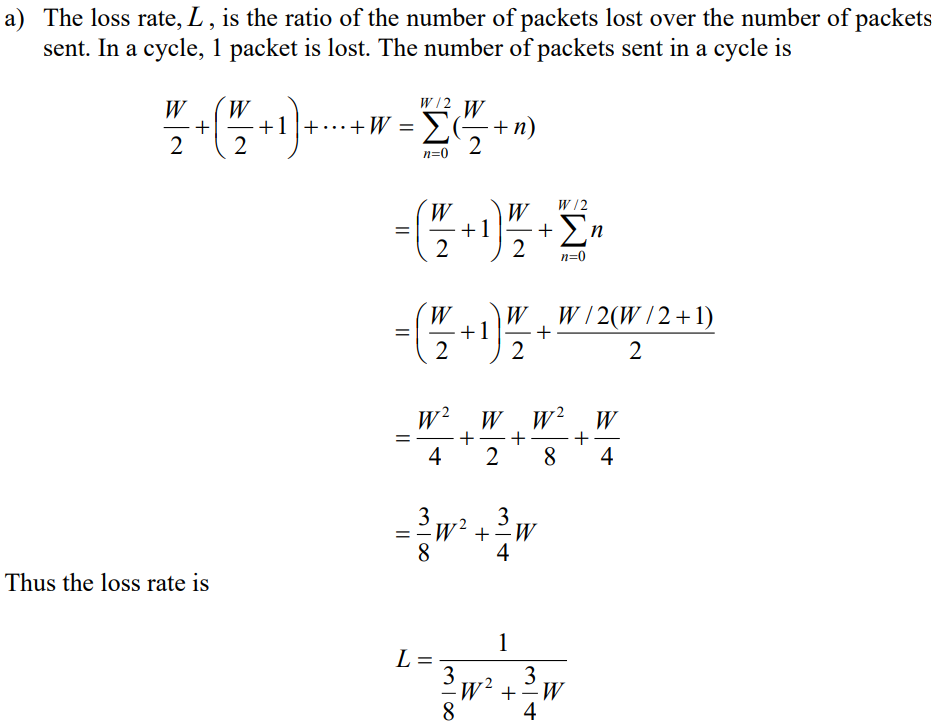
**25.** Suppose that instead of a multiplicative decrease, TCP decreased the window size by a constant amount. Would the resulting AIAD algorithm converge to an equal share algorithm?

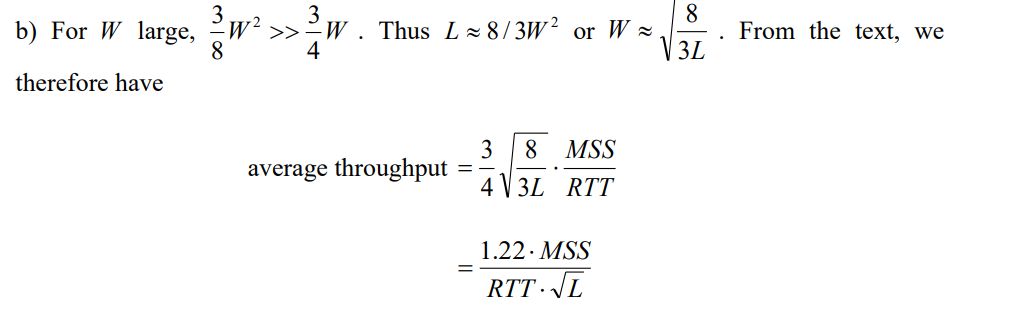


**26.** We discussed the doubling of the timeout interval after a timeout event. This mechanism is a form of congestion control. Why does TCP need a window-based congestion-control mechanism in addition to this doubling-timeout interval mechanism?

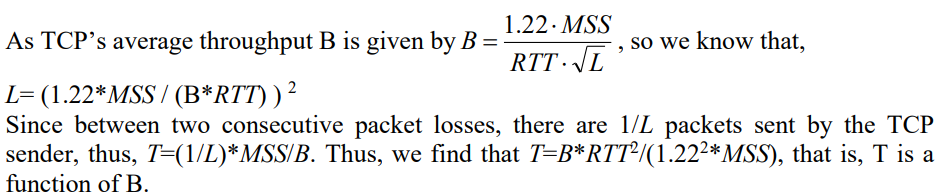
If TCP were a stop-and-wait protocol, then the doubling of the time out interval would suffice as a congestion control mechanism. However, TCP uses pipelining (and is therefore not a stop-and-wait protocol), which allows the sender to have multiple outstanding unacknowledged segments. The doubling of the timeout interval does not prevent a TCP sender from sending a large number of first-time-transmitted packets into the network, even when the end-to-end path is highly congested. Therefore a congestioncontrol mechanism is needed to stem the flow of “data received from the application above” when there are signs of network congestion.

**27.** Recall the macroscopic description of TCP throughput. In the period of time from when the S=10⋅R. time=6 RTT connection’s rate varies from W/(2 · RTT) to W/RTT, only one packet is lost (at the very end of the period). a. Show that the loss rate (fraction of packets lost) is equal to b. Use the result above to show that if a connection has loss rate L, then its average rate is approximately given by 1.22⋅MSSRTTL.





**28.** Let T (measured by RTT) denote the time interval that a TCP connection takes to increase its congestion window size from W/2 to W, where W is the maximum congestion window size. Argue that T is a function of TCP’s average throughput.



**29.** We implicitly assumed that the TCP sender always had data to send. Consider now the case that the TCP sender sends a large amount of data and then goes idle (since it has no more data to send) at t1. TCP remains idle for a relatively long period of time and then wants to send more data at t2. What are the advantages and disadvantages of having TCP use the cwnd and ssthresh values from t when starting to send data at t2? What alternative would you recommend? Why?

An advantage of using the earlier values of cwnd and ssthresh at t2 is that TCP would not have to go through slow start and congestion avoidance to ramp up to the throughput value obtained at t1. A disadvantage of using these values is that they may be no longer accurate. In particular, if the path has become more congested between t1 and t2, the sender will send a large window’s worth of segments into an already (more) congested path.

**30.** Why is it that voice and video traffic is often sent over TCP rather than UDP in today’s Internet?

Since most firewalls are configured to block UDP traffic, using TCP for video and voice traffic lets the traffic though the firewalls.

**31.** Suppose a process in Host C has a UDP socket with port number 6789. Suppose both Host A and Host B each send a UDP segment to Host C with destination port number 6789. Will both of these segments be directed to the same socket at Host C? If so, how will the process at Host C know that these two segments originated from two different hosts?

Yes, both segments will be directed to the same socket. For each received segment, at the socket interface, the operating system will provide the process with the IP addresses to determine the origins of the individual segments.

**32.** What is meant by the “match plus action” operation of a router or switch? In the case of destination-based forwarding packet switch, what is matched and what is the action taken? In the case of an SDN, name three fields that can be matched, and three actions that can be taken.

“Match plus action” means that a router or a switch tries to find a match between some of the header values of a packet with some entry in a flow table, and then based on that match, the router decides to which interface(s) the packet will be forwarded and even some more operations on the packet. In the case of destination-based forwarding packet switch, a router only tries to find a match between a flow table entry with the destination IP address of an arriving packet, and the action is to decide to which interface(s) the packet will be forwarded. In the case of an SDN, there are many fields can be matched, for example, IP source address, TCP source port, and source MAC address; there are also many actions can be taken, for example, forwarding, dropping, and modifying a field value.

**33.** What is meant by a “plug-and-play” or “zeroconf” protocol?

A plug-and-play or zeroconf protocol means that the protocol is able to automatically configure a host’s network-related aspects in order to connect the host into a network.

**34.** Discuss why each input port in a high-speed router stores a shadow copy of the forwarding table.

With the shadow copy, the forwarding lookup is made locally, at each input port, without invoking the centralized routing processor. Such a decentralized approach avoids creating a lookup processing bottleneck at a single point within the router.

**35.** What is meant by a control plane that is based on per-router control? In such cases, when we say the network control and data planes are implemented “monolithically,” what do we mean?

Per-router control means that a routing algorithm runs in each and every router; both forwarding and routing function are constrained within each router. Each router has a routing component that communicates with the routing components in other routers to compute the values for its forwarding table. In such cases, we say that the network control and data planes are implemented monolithically because each router works as an independent entity that implements its own control and data planes.

**36.**Compare and contrast the properties of a centralized and a distributed routing algorithm. Give an example of a routing protocol that takes a centralized and a decentralized approach.

A centralized routing algorithm computes the least-cost path between a source and destination by using complete, global knowledge about the network. The algorithm needs to have the complete knowledge of the connectivity between all nodes and all links’ costs. The actual calculation can be run at one site or could be replicated in the routing component of each and every router. A distributed routing algorithm calculates the lease-cost path in an iterative, distributed manner by the routers. With a decentralized algorithm, no node has the complete information about the costs of all network links. Each node begins with only the knowledge of the costs of its own directly attached links, and then through an iterative process of calculation and information exchange with its neighboring nodes, a node gradually calculates the least-cost path to a destination or a set of destinations. OSPF protocol is an example of centralized routing algorithm, and BGP is an example of a distributed routing algorithm.

**37.**What is the “count to infinity” problem in distance vector routing?

The count-to-infinity problem refers to a problem of distance vector routing. The problem means that it takes a long time for a distance vector routing algorithm to converge when there is a link cost increase. For example, consider a network of three nodes x, y, and z. Suppose initially the link costs are c(x,y)=4, c(x,z)=50, and c(y,z)=1. The result of distance-vector routing algorithm says that z’s path to x is z→y→ x and the cost is 5(=4+1). When the cost of link (x,y) increases from 4 to 60, it will take 44 iterations of running the distance-vector routing algorithm for node z to realize that its new least-cost path to x is via its direct link to x, and hence y will also realize its least-cost path to x is via z.

**38.**Consider a general topology (that is, not the specific network shown above) and a synchronous version of the distance-vector algorithm. Suppose that at each iteration, a node exchanges its distance vectors with its neighbors and receives their distance vectors. Assuming that the algorithm begins with each node knowing only the costs to its immediate neighbors, what is the maximum number of iterations required before the distributed algorithm converges?

At each iteration, a node exchanges distance tables with its neighbors. Thus, if you are node A, and your neighbor is B, all of B's neighbors (which will all be one or two hops from you) will know the shortest cost path of one or two hops to you after one iteration (i.e., after B tells them its cost to you). Let d be the “diameter” of the network - the length of the longest path without loops between any two nodes in the network. Using the reasoning above, after d −1 iterations, all nodes will know the shortest path cost of d or fewer hops to all other nodes. Since any path with greater than d hops will have loops (and thus have a greater cost than that path with the loops removed), the algorithm will converge in at most d −1 iterations. ASIDE: if the DV algorithm is run as a result of a change in link costs, there is no a priori bound on the number of iterations required until convergence unless one also specifies a bound on link costs.

**39.**Consider the count-to-infinity problem in the distance vector routing. Will the count-to-infinity problem occur if we decrease the cost of a link? Why? How about if we connect two nodes which do not have a link?

NO, this is because that decreasing link cost won’t cause a loop (caused by the next-hop relation of between two nodes of that link). Connecting two nodes with a link is equivalent to decreasing the link weight from infinite to the finite weight.

**40.**Argue that for the distance-vector algorithm in Figure 5.6 , each value in the distance vector D(x) is non-increasing and will eventually stabilize in a finite number of steps.

At each step, each updating of a node’s distance vectors is based on the Bellman-Ford equation, i.e., only decreasing those values in its distance vector. There is no increasing in values. If no updating, then no message will be sent out. Thus, D(x) is non-increasing. Since those costs are finite, then eventually distance vectors will be stabilized in finite steps.

**41.**What is meant by an area in an OSPF autonomous system? Why was the concept of an area introduced?

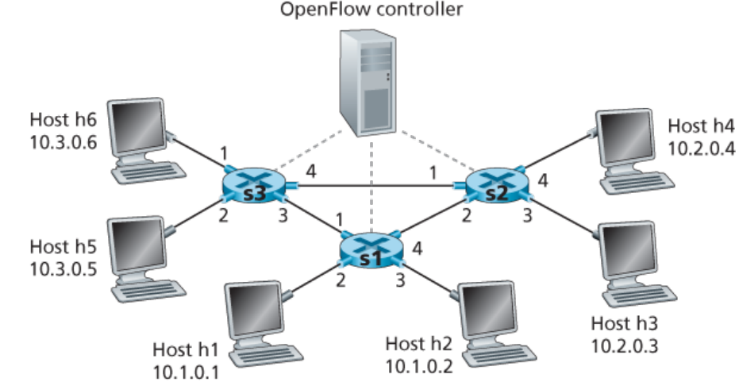
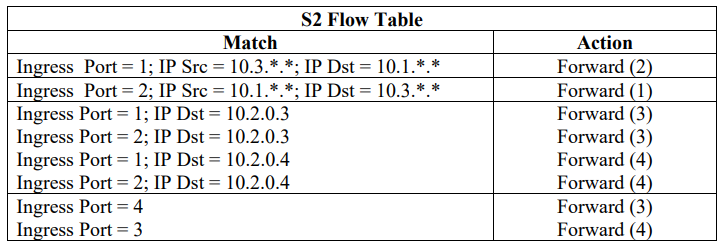
An area in an OSPF autonomous system is refers to a set of routers, in which each router broadcasts its link state to all other routers in the same set. An OSPF AS can be configured hierarchically into multiple areas, with each area running its own OSPF link-state routing algorithm. Within each area, one or more area border routers are responsible for routing packets outside the area. The concept of area is introduced for scalability reason, i.e., we would like to build a hierarchical routing for a large scale OSPF AS, and an area is an important building block in hierarchical routing.

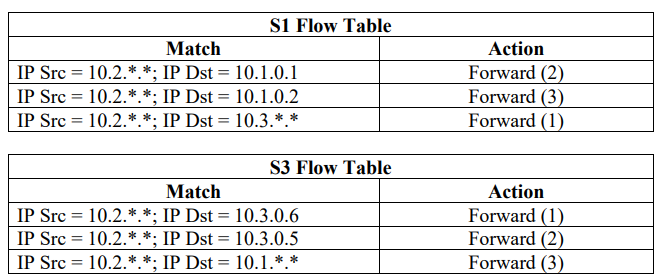
**42.** Define and contrast the following terms: subnet, prefix, and BGP route.

A subnet is a portion of a larger network; a subnet does not contain a router; its boundaries are defined by the router and host interfaces. A prefix is the network portion of a CDIRized address; it is written in the form a.b.c.d/x ; A prefix covers one or more subnets. When a router advertises a prefix across a BGP session, it includes with the prefix a number of BGP attributes. In BGP jargon, a prefix along with its attributes is a BGP route (or simply a route).

**43.**Name three header fields in an IP datagram that can be “matched” in OpenFlow 1.0 generalized forwarding. What are three IP datagram header fields that cannot be “matched” in OpenFlow?

Three example header fields in an IP datagram that can be matched in OpenFlow 1.0 generalized forwarding are IP source address, TCP source port, and source MAC address. Three fields that cannot be matched are: TTL field, datagram length field, header checksum (which depends on TTL field)

**44.**Consider the SDN OpenFlow network shown in Figure 4.30 . Suppose that the desired forwarding behavior for datagrams arriving at s2 is as follows: any datagrams arriving on input port 1 from hosts h5 or h6 that are destined to hosts h1 or h2 should be forwarded over output port 2; any datagrams arriving on input port 2 from hosts h1 or h2 that are destined to hosts h5 or h6 should be forwarded over output port 1; any arriving datagrams on input ports 1 or 2 and destined to hosts h3 or h4 should be delivered to the host specified; hosts h3 and h4 should be able to send datagrams to each other.

**45.**Consider again the scenario from P19 above. Give the flow tables entries at packet switches s1 and s3, such that any arriving datagrams with a source address of h3 or h4 are routed to the destination hosts specified in the destination address field in the IP datagram. (Hint: Your forwarding table rules should include the cases that an arriving datagram is destined for a directly attached host or should be forwarded to a neighboring router for eventual host delivery there.)

**46.** Describe the main role of the communication layer, the network-wide state-management layer, and the network-control application layer in an SDN controller.

The communication layer is responsible for the communication between the SDN controller and those controlled network devices, via a protocol such as OpenFlow. Through this layer, an SDN controller controls the operation of a remote SDNenabled switch, host, or other devices, and a device communicates locally-observed events (e.g., a message indicating a link failure) to the controller. The network-wide state-management layer provides up-to-date information about state a network’s hosts, links, switches, and other SDN-controlled devices. A controller also maintains a copy of the flow tables of the various controlled devices. The network-control application layer represents the brain of SDN control plane. The applications at this layer use the APIs provided by a SDN controller to specify and control the data plane in the network devices. For example, a routing network-control application might determine the end-end paths between sources and destinations. Another network application might perform access control.

**47.** Suppose you wanted to implement a new routing protocol in the SDN control plane. At which layer would you implement that protocol? Explain.

I would implement a new routing protocol at the SDN’s network-control application layer, as this is the layer where a routing protocol determines the end-to-end paths between sources and destinations.

**48.**What types of messages flow across an SDN controller’s northbound and southbound APIs? Who is the recipient of these messages sent from the controller across the southbound interface, and who sends messages to the controller across the northbound interface?

The following is a list of types of messages flow across a SDN controller’s southbound from the controller to the controlled devices. The recipient of these messages is a controlled packet switch. • Configuration. This message allows the controller to query and set a switch’s configuration parameters. • Modify-state. This message is used by a controller to add/delete or modify entries in the switch’s flow table, and to set switch port properties. • Read-state. This message is used by a controller to collect statistics and counter values from the switch’s flow table and ports. • Send-packet. This message is used by the controller to send a specific packet out of a specified port at the controlled switch.

**49.**Describe the purpose of two types of OpenFlow messages (of your choosing) that are sent from a controlled device to the controller. Describe the purpose of two types of Openflow messages (of your choosing) that are send from the controller to a controlled device

Two types of messages from a controlled device to a controller: • Flow-removed message. Its purpose is to inform the controller that a flow table entry has been removed, for example, by a timeout or as the result of a received modify-state message. • Port-status message. Its purpose is to inform the controller of a change in port status. Two types of messages from a controller to a controlled device: • Modify-state. The purpose is to add/delete or modify entries in the switch’s flow table, and to set switch port properties. • Read-state. The purpose is to collect statistics and counter values rom the switch’s flow table and ports.

**50.** What is the purpose of the service abstraction layer in the OpenDaylight SDN controller?

The service abstraction layer allows internal network service applications to communicate with each other. It allows controller components and applications to invoke each other’s services and to subscribe to events they generate. This layer also provides a uniform abstract interface to the specific underlying communications protocols in the communication layer, including OpenFlow and SNMP.

**51.** Name four different types of ICMP messages.

Echo reply (to ping), type 0, code 0 Destination network unreachable, type 3 code 0 Destination host unreachable, type 3, code 1. Source quench (congestion control), type 4 code 0.

**52.**What two types of ICMP messages are received at the sending host executing the Traceroute program?

ICMP warning message (type 11 code 0) and a destination port unreachable ICMP message (type 3 code 3).

**53.**Define the following terms in the context of SNMP: managing server, managed device, network management agent and MIB.

A managing server is an application, typically with a human in the loop, running in a centralized network management station in a network operation center. It controls the collection, processing, analysis, and/or display of network management information. Actions are initiated in a managing server to control network behavior and a network administrator uses a managing server to interact with the network’s devices. A managed device is a piece of network equipment (including its software) that resides on a managed network. A managed device might be a host, router, switch, middlebox, modem, thermometer, or other network-connected device. A network management agent is a process running in a managed device that communicates with a managing server, taking local actions at the managed device under the command and control of the managing server. Management Information Base (MIB) collects the information associated with those managed objects in a managed network. A MIB object might be a counter, such as the number of IP datagrams discarded at a router due to errors in an IP datagram header, or the number of UDP segments received at a host, or the status information such as whether a particular device is functioning correctly.

**54.** What are the purposes of the SNMP GetRequest and SetRequest messages?

GetRequest is a message sent from a managing server to an agent to request the value of one or more MIB objects at the agent’s managed device. SetRequest is a message used by a managing server to set the value of one or more MIB objects in a managed device.

**55.** What is the purpose of the SNMP trap message?

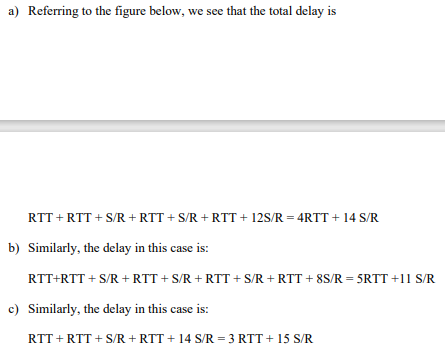
A SNMP trap message is generated as a response to an event happened on a managed device for which the device’s managing server requires notification. It is used for notifying a managing server of an exceptional situation (e.g., a link interface going up or down) that has resulted in changes to MIB object values.

**56.**In this problem, we consider the delay introduced by the TCP slow-start phase. Consider a client and a Web server directly connected by one link of rate R. Suppose the client wants to retrieve an object whose size is exactly equal to 15 S, where S is the maximum segment size (MSS). Denote the round-trip time between client and server as RTT (assumed to be constant). Ignoring protocol headers, determine the time to retrieve the object (including TCP connection establishment) when a. b. c. . 4 S/R>S/R+RTT>2S/R

S/R+RTT>4 S/R

S/R>RTT

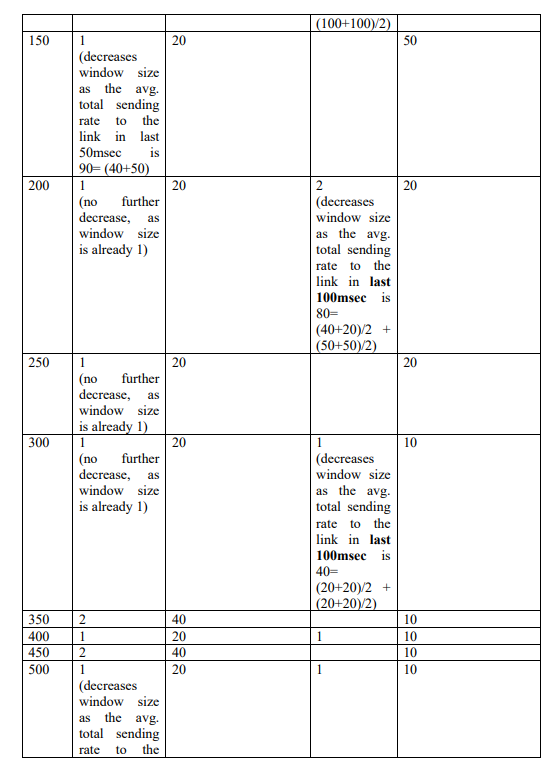
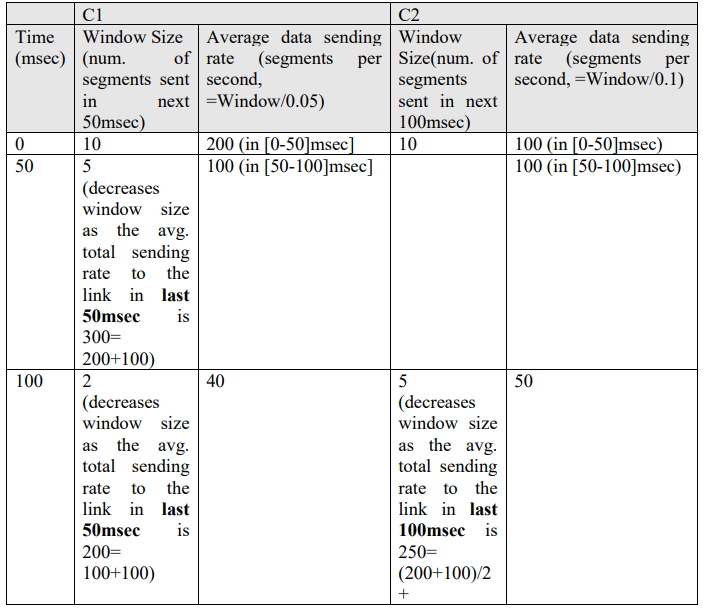
Referring to the figure below, we see that the total delay is:



**57.**Consider the two ways in which communication occurs between a managing entity and a managed device: request-response mode and trapping. What are the pros and cons of these two approaches, in terms of (1) overhead, (2) notification time when exceptional events occur, and (3) robustness with respect to lost messages between the managing entity and the device?

Request response mode will generally have more overhead (measured in terms of the number of messages exchanged) for several reasons. First, each piece of information received by the manager requires two messages: the poll and the response. Trapping generates only a single message to the sender. If the manager really only wants to be notified when a condition occurs, polling has more overhead, since many of the polling messages may indicate that the waited-for condition has not yet occurred. Trapping generates a message only when the condition occurs. Trapping will also immediately notify the manager when an event occurs. With polling, the manager needs will need to wait for half a polling cycle (on average) between when the event occurs and the manager discovers (via its poll message) that the event has occurred. If a trap message is lost, the managed device will not send another copy. If a poll message, or its response, is lost the manager would know there has been a lost message (since the reply never arrives). Hence the manager could repoll, if needed.

**58.** Consider a simplified TCP’s AIMD algorithm where the congestion window size is measured in number of segments, not in bytes. In additive increase, the congestion window size increases by one segment in each RTT. In multiplicative decrease, the congestion window size decreases by half (if the result is not an integer, round down to the nearest integer). Suppose that two TCP connections, C1 and C2 , share a single congested link of speed 30 segments per second. Assume that both C and C are in the congestion avoidance phase. Connection C ’s RTT is 50 msec and connection C ’s RTT is 100 msec. Assume that when the data rate in link exceeds the link’s speed, all TCP connections experience data segment loss. a. If both C1 and C2 at time t0 have a congestion window of 10 segments, what are their congestion window sizes after 1000 msec? b. In the long run, will these two connections get the same share of the bandwidth of the congested link? Explain.

****