Prof. Athina Markopoulou
Spring 2016

Intro to Computer Networks
EECS148/CompSci132

Homework 4 Solution
Topic : Network Layer

1. *(20 Points)* **IP Addressing.**

   (a) **Ch. 4, Problem 13.**
   Consider a router that interconnects three subnets: Subnet I, Subnet 2, and Subnet 3. Suppose all of the interfaces in each of these three subnets are required to have the prefix 223.1 .17/24. Also suppose that Subnet I is required to support up to 63 interfaces, Subnet 2 is to support up to 95 interfaces, and Subnet 3 is to support up to 16 interfaces. Provide three network addresses (of the form a.b.c .d/x) that satisfy these constraints.
   Solution:
   There are many choices that satisfy these constraints (the three prefixes can fit the number of addresses in each subnet, and there should be no overlap between addresses in different subnets). Here is one such choice:
   - Subnet 2 : 223.1.17.0/26
   - Subnet 1 : 223.1.17.128/25
   - Subent 3 : 223.1.17.192/28

   (b) **Ch. 4, Problem 21.**
   Consider the network setup in Figure 4.22. Suppose that the ISP instead assigns the router the address 24.34.112.235 and that the network address of the home network is 192.168. 1124. a. Assign addresses to all interfaces in the home network. b. Suppose each host has two ongoing TCP connections, all to port 80 at host 128.119.40.86. Provide the six corresponding entries in the NAT translation table.

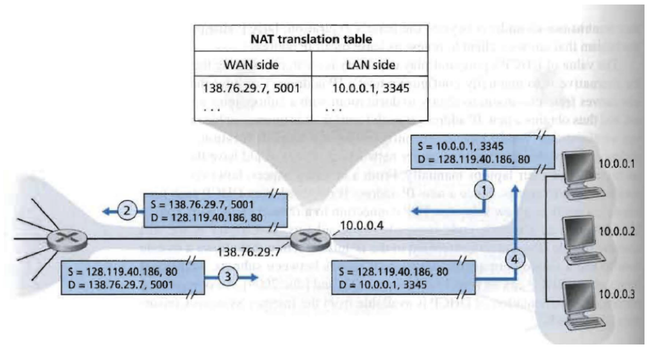


Figure 1: Figure 4.22 in the book (repeated here for convenience)

Solution:

i. Host interfaces: 192.168.1.1, 192.168.1.2, 192.168.1.3. Router interface: 192.168.1.4.

ii. NAT translation table:

WAN Side	LAN Side
24.34.112.235, 4000	192.168.1.1, 3345
24.34.112.235, 4001	192.168.1.1, 3346
24.34.112.235, 4002	192.168.1.2, 3445
24.34.112.235, 4003	192.168.1.2, 3446
24.34.112.235, 4004	192.168.1.3, 3545
24.34.112.235, 4005	192.168.1.3, 3546

2. *(20 Points)* **BGP (Ch. 4, Problem 40)**

In Figure 4.42, consider the path information that reaches stub networks W, X, and Y. Based on the information available at W and X, what are their respective views of the network topology? Justify your answer. The topology view at Y is shown below.

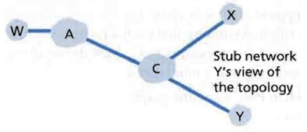
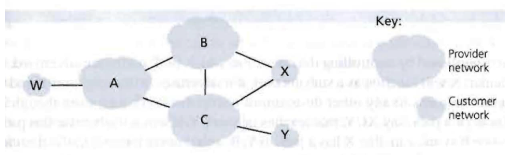


Figure 2: The view from Y



Solution:

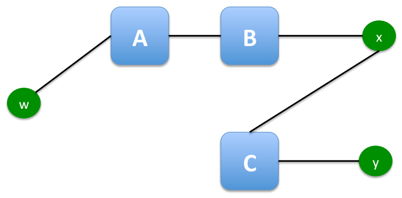


Figure 4: X's view of the topology

In Fig. 4 and Fig. 5 , X does not know about the AC link since X does not receive an advertised route to w or to y that contain the AC link (i.e., X receives no advertisement containing both AS A and AS C on the path to a destination.

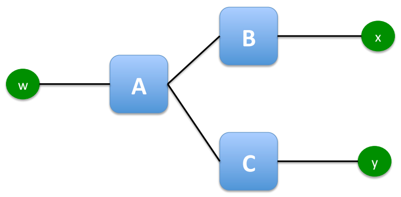
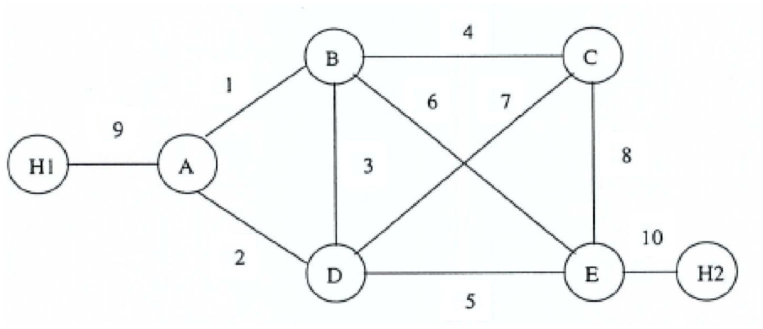


Figure 5: W's view of the topology

3. *(60 Points)* **Routing Algorithms.**

   (a) The following figure shows a network with 2 hosts (H_1 and H_2), 5 routers (A, B, C, D, E).



There are 10 bi-directional links, labeled from 1 to 10, with their transmission bandwidth and propagation delay as follows:

Link #	Trans. BW (100 kb/s)	Propagation delay (ms)
1	4	5
2	1	30
3	1	20
4	2	10
5	1	20
6	0.5	40
7	5	6
8	0.5	20
9	4	5
10	4	5

Packets are sent in the network in a stored-and-forward manner. Suppose that the network is initially empty and processing delay is negligible, *i.e.*, the only delays relevant here are packet transmission time and propagation delay. Now H_1 wants to send a packet of 2000 bits to H_2.

(a) *(20 Points)* Use Dijkstra's algorithm to determine the route the packet should take in order to achieve minimal delay. Show your work on Fig. 7.

(b) *(10 Points)* Is this route the same as the route with the minimum number of hops?

(c) *(20 Points)* Now suppose that the network employs the Bellman-Ford distributed routing algorithm (to find the path of minimum delay for each 2000bits packet). Fill in Figure 8 to show the exchanges of routing tables until the final result. (You may ignore the entries for hosts H_1 and H_2, and focus only on the routers.) In each router's table, the routers in parenthesis are its neighbors.

(d) *(10 Points)* Now assume that links AB and BE fail. Run Bellman-Ford algorithm, starting from the steady state achieved in question (c) above. Show your calculations similarly to (or on a copy of) Fig. 8. Does the algorithm converge?

Solution:

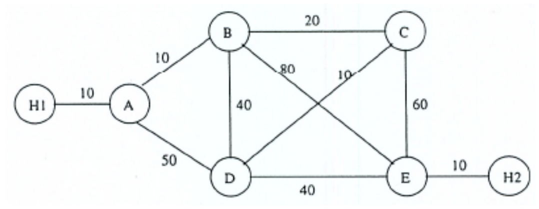


Figure 6: Network topology showing the delay in ms in each link

(a) Figure 6 shows the total delay (in ms) in a link for the 2 kb packet if it were traveling on that link. Figure **??** shows the Dijkstra algorithm. Note that we did not show the hosts. We see that the minimum delay from H_1 to H_2 is 100 ms and the route is $\{H_1, A, B, C, D, E, H_2\}$.

(b) The route to achieve minimal delay found above is different from the route obtained with minimum hops, *e.g.*, $\{H_1, A, D, E, H_2\}$.

(c) Figure **??** shows the exchanges according to Bellman-Ford algorithm. Note that an entry in a node would need to be updated only if the corresponding entries of its neighbors or itself change compared with the last exchange.

The total delay calculated is again 100 ms, and the route is $\{H_1, A, B, C, D, E, H_2\}$.

(d) Figure 9 shows the exchanges according to Bellman-Ford algorithm after the failure of AB and BE. The algorithm converges.

The new total delay is 110 ms, and the route is $\{H_1, A, D, E, H_2\}$.

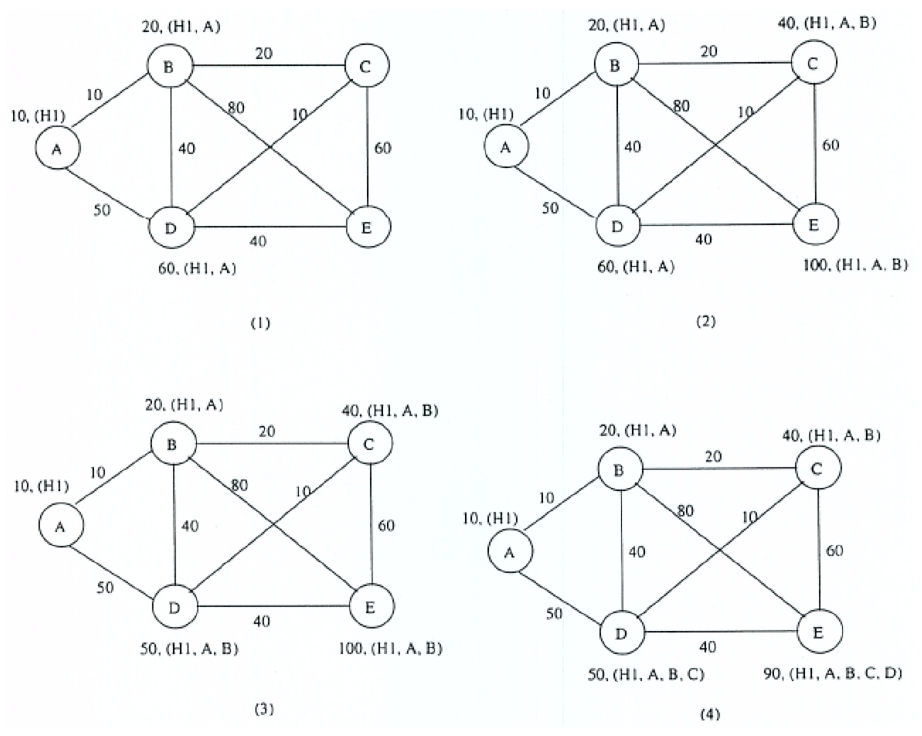


Figure 7: Dijkstra's shortest path algorithm

4. *(Optional +50 Points)* **Programming Assignment.** In this programming assignment you will be writing a distributed set of procedures that implement a distributed asynchronous distance-vector routing protocol for the small network shown in the figure (p.429 in v6 of your book). The full programming assignment can be found in the following URL:

   `http://media.pearsoncmg.com/aw/aw_kurose_network_3/labs/lab6/lab6.html`

   You are only asked to do only the **basic** assignment. You can do it in C or java version, or you can skip it without penalty(it is optional).

   Instruction:

   Structure

   In the basic part of this assignment, you will be writing a "distributed" set of procedures that implement a distributed asynchronous distance vector routing for the network shown in Figure10. You are to write the procedures rtinit0(), rtinit1(), rtinit2(), rtinit3() and rtupdate0(), rtupdate1(), rtupdate2(), rtupdate3() which together will implement a distributed, asynchronous computation of the distance tables for the topology and costs shown in Figure10.

   Remember: You should put your procedures for nodes 0 through 3 in files called node0.c, ... node3.c. You are NOT allowed to declare any global variables that are visible outside of a given C file (e.g., any global variables you define in node0.c. may only be accessed inside node0.c). This is to force you to abide by the coding conventions that you would have to adopt is you were really running the procedures in four distinct nodes.

Initial State

Node A (B,D)		
Dest	Via	Delay
A	A	0
B	-	∞
C	-	∞
D	-	∞
E	-	∞

Node B (A,C,D,E)		
Dest	Via	Delay
A	-	∞
B	B	0
C	-	∞
D	-	∞
E	-	∞

Node C (B,D,E)		
Dest	Via	Delay
A	-	∞
B	-	∞
C	C	0
D	-	∞
E	-	∞

Node D (A,B,C,E)		
Dest	Via	Delay
A	-	∞
B	-	∞
C	-	∞
D	D	0
E	-	∞

Node E (B,C,D)		
Dest	Via	Delay
A	-	∞
B	-	∞
C	-	∞
D	-	∞
E	E	0

Tables after first exchange

Node A (B,D)		
Dest	Via	Delay
A	A	0
B	B	10*
C	-	∞
D	D	50*
E	-	∞

Node B (A,C,D,E)		
Dest	Via	Delay
A	A	10*
B	B	0
C	C	20*
D	D	40*
E	E	80*

Node C (B,D,E)		
Dest	Via	Delay
A	-	∞
B	B	20*
C	C	0
D	D	10*
E	E	60*

Node D (A,B,C,E)		
Dest	Via	Delay
A	A	50*
B	B	40*
C	C	10*
D	D	0
E	E	40*

Node E (B,C,D)		
Dest	Via	Delay
A	-	∞
B	B	80*
C	C	60*
D	D	40*
E	E	0

Tables after second exchange

Node A (B,D)		
Dest	Via	Delay
A	A	0
B	B	10
C	B	30*
D	D	50
E	B	90*

Node B (A,C,D,E)		
Dest	Via	Delay
A	A	10
B	B	0
C	C	20
D	C	30*
E	E	80

Node C (B,D,E)		
Dest	Via	Delay
A	B	30*
B	B	20
C	C	0
D	D	10
E	D	50*

Node D (A,B,C,E)		
Dest	Via	Delay
A	A	50
B	C	30*
C	C	10
D	D	0
E	E	40

Node E (B,C,D)		
Dest	Via	Delay
A	B	90*
B	B	80
C	D	50*
D	D	40
E	E	0

Tables after third exchange

Node A (B,D)		
Dest	Via	Delay
A	A	0
B	B	10
C	B	30
D	B	40*
E	B	90

Node B (A,C,D,E)		
Dest	Via	Delay
A	A	10
B	B	0
C	C	20
D	C	30
E	C	70

Node C (B,D,E)		
Dest	Via	Delay
A	B	30
B	B	20
C	C	0
D	D	10
E	D	50

Node D (A,B,C,E)		
Dest	Via	Delay
A	C	40*
B	C	30
C	C	10
D	D	0
E	E	40

Node E (B,C,D)		
Dest	Via	Delay
A	B	90
B	D	70*
C	D	50
D	D	40
E	E	0

Tables after fourth exchange

Node A (B,D)		
Dest	Via	Delay
A	A	0
B	B	10
C	B	30
D	B	40
E	B	90

Node B (A,C,D,E)		
Dest	Via	Delay
A	A	10
B	B	0
C	C	20
D	C	30
E	C	70

Node C (B,D,E)		
Dest	Via	Delay
A	B	30
B	B	20
C	C	0
D	D	10
E	D	50

Node D (A,B,C,E)		
Dest	Via	Delay
A	C	40
B	C	30
C	C	10
D	D	0
E	E	40

Node E (B,C,D)		
Dest	Via	Delay
A	B	80*
B	D	70
C	D	50
D	D	40
E	E	0

Tables after fifth exchange

Node A (B,D)		
Dest	Via	Delay
A	A	0
B	B	10
C	B	30
D	B	40
E	B	90

Node B (A,C,D,E)		
Dest	Via	Delay
A	A	10
B	B	0
C	C	20
D	C	30
E	C	70

Node C (B,D,E)		
Dest	Via	Delay
A	B	30
B	B	20
C	C	0
D	D	10
E	D	50

Node D (A,B,C,E)		
Dest	Via	Delay
A	C	40
B	C	30
C	C	10
D	D	0
E	E	40

Node E (B,C,D)		
Dest	Via	Delay
A	B	80
B	D	70
C	D	50
D	D	40
E	E	0

* : changes

No more updates – Algorithm converges

Figure 8: Exchange of tables using Bellman-Ford algorithm

Steady State

Node A (B,D)			Node B (A,C,D,E)			Node C (B,D,E)			Node D (A,B,C,E)			Node E (B,C,D)		
Dest	Via	Delay	Dest	Via	Delay	Dest	Via	Delay	Dest	Via	Delay	Dest	Via	Delay
A	A	0	A	A	10	A	B	30	A	C	40	A	B	80
B	B	10	B	B	0	B	B	20	B	C	30	B	D	70
C	B	30	C	C	20	C	C	0	C	C	10	C	D	50
D	B	40	D	C	30	D	D	10	D	D	0	D	D	40
E	B	90	E	C	70	E	D	50	E	E	40	E	E	0

Table after first exchange following AB's and BE's failures

Node A (D)			Node B (C,D)			Node C (B,D,E)			Node D (A,B,C,E)			Node E (C,D)		
Dest	Via	Delay	Dest	Via	Delay	Dest	Via	Delay	Dest	Via	Delay	Dest	Via	Delay
A	A	0	A	C	50*	A	B	30	A	C	40	A	D	80*
B	D	80*	B	B	0	B	B	20	B	C	30	B	D	70
C	D	60*	C	C	20	C	C	0	C	C	10	C	D	50
D	D	50*	D	C	30	D	D	10	D	D	0	D	D	40
E	D	90*	E	C	70	E	D	50	E	E	40	E	E	0

Table after second exchange

Node A (D)			Node B (C,D)			Node C (B,D,E)			Node D (A,B,C,E)			Node E (C,D)		
Dest	Via	Delay	Dest	Via	Delay	Dest	Via	Delay	Dest	Via	Delay	Dest	Via	Delay
A	A	0	A	C	50	A	D	50*	A	C	40	A	D	80
B	D	80	B	B	0	B	B	20	B	C	30	B	D	70
C	D	60	C	C	20	C	C	0	C	C	10	C	D	50
D	D	50	D	C	30	D	D	10	D	D	0	D	D	40
E	D	90	E	C	70	E	D	50	E	E	40	E	E	0

Table after third exchange

Node A (D)			Node B (C,D)			Node C (B,D,E)			Node D (A,B,C,E)			Node E (C,D)		
Dest	Via	Delay	Dest	Via	Delay	Dest	Via	Delay	Dest	Via	Delay	Dest	Via	Delay
A	A	0	A	C	70*	A	D	50	A	A	50*	A	D	80
B	D	80	B	B	0	B	B	20	B	C	30	B	D	70
C	D	60	C	C	20	C	C	0	C	C	10	C	D	50
D	D	50	D	C	30	D	D	10	D	D	0	D	D	40
E	D	90	E	C	70	E	D	50	E	E	40	E	E	0

Table after fourth exchange

Node A (D)			Node B (C,D)			Node C (B,D,E)			Node D (A,B,C,E)			Node E (C,D)		
Dest	Via	Delay	Dest	Via	Delay	Dest	Via	Delay	Dest	Via	Delay	Dest	Via	Delay
A	A	0	A	C	70	A	D	60*	A	A	50	A	D	90*
B	D	80	B	B	0	B	B	20	B	C	30	B	D	70
C	D	60	C	C	20	C	C	0	C	C	10	C	D	50
D	D	50	D	C	30	D	D	10	D	D	0	D	D	40
E	D	90	E	C	70	E	D	50	E	E	40	E	E	0

Table after fifth exchange

Node A (D)			Node B (C,D)			Node C (B,D,E)			Node D (A,B,C,E)			Node E (C,D)		
Dest	Via	Delay	Dest	Via	Delay	Dest	Via	Delay	Dest	Via	Delay	Dest	Via	Delay
A	A	0	A	C	80*	A	D	60	A	A	50	A	D	90
B	D	80	B	B	0	B	B	20	B	C	30	B	D	70
C	D	60	C	C	20	C	C	0	C	C	10	C	D	50
D	D	50	D	C	30	D	D	10	D	D	0	D	D	40
E	D	90	E	C	70	E	D	50	E	E	40	E	E	0

Table after sixth exchange

Node A (D)			Node B (C,D)			Node C (B,D,E)			Node D (A,B,C,E)			Node E (C,D)		
Dest	Via	Delay	Dest	Via	Delay	Dest	Via	Delay	Dest	Via	Delay	Dest	Via	Delay
A	A	0	A	C	80	A	D	60	A	A	50	A	D	90
B	D	80	B	B	0	B	B	20	B	C	30	B	D	70
C	D	60	C	C	20	C	C	0	C	C	10	C	D	50
D	D	50	D	C	30	D	D	10	D	D	0	D	D	40
E	D	90	E	C	70	E	D	50	E	E	40	E	E	0

*: changes

No more updates – Algorithm converges

Figure 9: Exchange of tables using Bellman-Ford algorithm after AB and BE fail

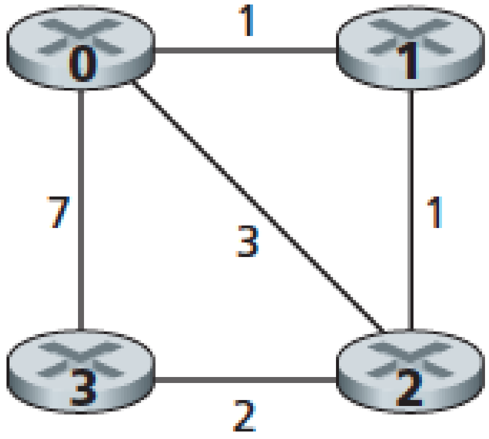


Figure 10: Structure of the network

You can find the preototype version of files needed for this assignment here:**prog3.c**, **node0.c**, **node1.c**, **node2.c**, **node3.c**

Routines

For ever node**X** in the network you have to implement two routines:rtinit**X**() and rtupdate**X**(struct rtpkt *rcvdpkt). Routine rtinit() will do initialization of distance table and then send it to directly-connected neighbors. Routine rtupdate() is the "heart" of the distance vector algorithm. This routine is responsible for updating the distance table and notifying the neighbors. Bellow, we'll describe both routines for node0(rtinit0() and rtupdate0()) in further details. Similar routines are defined for nodes 1, 2 and 3. Thus, you will write 8 procedures in all: rtinit0(), rtinit1(), rtinit2(), rtinit3(),rtupdate0(), rtupdate1(), rtupdate2(), rtupdate3().

rtinit0() This routine will be called once at the beginning of the emulation. rtinit0() has no arguments. It should initialize the distance table in node 0 to reflect the direct costs of 1, 3, and 7 to nodes 1, 2, and 3, respectively. In Figure 1, all links are bi-directional and the costs in both directions are identical. After initializing the distance table, and any other data structures needed by your node 0 routines, it should then send its directly-connected neighbors (in this case, 1, 2 and 3) the cost of it minimum cost paths to all other network nodes. This minimum cost information is sent to neighboring nodes in a routing packet by calling the routine tolayer2(), as described below. The format of the routing packet is also described below.

rtupdate0(struct rtpkt *rcvdpkt). This routine will be called when node 0 receives a routing packet that was sent to it by one if its directly connected neighbors. The parameter *rcvdpkt is a pointer to the packet that was received. The values it receives in a routing packet from some other node i contain i's current shortest path costs to all other network nodes. rtupdate0() uses these received values to update its own distance table (as specified by the distance vector algorithm). If its own minimum cost to another node changes as a result of the update, node 0 informs its directly connected neighbors of this change in minimum cost by sending them a routing packet. Recall that in the distance vector algorithm, only directly connected nodes will exchange routing packets. Thus nodes 1 and 2 will communicate with each other, but nodes 1 and 3 will node communicate with each other.

Simulated Network Enviorment

Your procedures rtinit0(), rtinit1(), rtinit2(), rtinit3() and rtupdate0(), rtupdate1(), rtupdate2(), rtupdate3() send routing packets (whose format is described above) into the medium. The medium will deliver packets in-order, and without loss to the specified destination. Only directly-connected nodes can communicate. The delay between is sender and receiver is variable (and unknown).

When you compile your procedures and my procedures together and run the resulting program, you will be asked to specify value of tracing variable. This will be explained in output section.

Packet format

As we saw in class, the distance table inside each node is the principal data structure used by the distance vector algorithm. You will find it convenient to declare the distance table as a 4-by-4 array of int's, where entry [i,j] in the distance table in node 0 is node 0's currently computed cost to node i via direct neighbor j. If 0 is not directly connected to j, you can ignore this entry. We will use the convention that the integer value 999 is "infinity."

Output

For your sample output, your procedures should print out a message whenever your rtinit0(), rtinit1(), rtinit2(), rtinit3() or rtupdate0(), rtupdate1(), rtupdate2(), rtupdate3() procedures are called, giving the time (available via my global variable clocktime). For rtupdate0(), rtupdate1(), rtupdate2(), rtupdate3() you should print the identity of the sender of the routing packet that is being passed to your routine, whether or not the distance table is updated, the contents of the distance table (you can use my pretty-print routines), and a description of any messages sent to neighboring nodes as a result of any distance table updates.

The sample output should be an output listing with a TRACE value of 2. Highlight the final distance table produced in each node. Your program will run until there are no more routing packets in-transit in the network, at which point our emulator will terminate.