Q2 – the nature of congestion

a. במקרה שלכל החבילות יש את אותו הvalue אבל slack שונה, עדיף להשתמש ב**EDF** מאחר והEDF עובד כך – במידה והחבילה שצריכה להיכנס לתור היא עם slack גבוה יותר מאחת החבילות שיש כבר בתור, אז מוציאים מהתור את החבילה עם הslack המינימלי ומכניסים את החבילה עם הslack הגבוה יותר. בנוסף בעת הprocessing שולחים את החבילה עם ה slack המינימלי. כך בעצם נוכל למקסם את הטיפול והשליחה של החבילות בלי שיגמר הslack של החבילה.

לכן במידה ו value של כל החבילות שווה אך הslack שונה עדיף להשתמש בEDF אשר דואג להכניס לתור את החבילות עם הסבירות הכי גבוהה שנצליח לטפל בהם לפני שהslack נגמר.

b. במקרה שלכל החבילות יש את אותו הslack אבל value שונה, עדיף להשתמש ב**bound delay** מאחר והbound delay עובד כך – במידה והחבילה שצריכה להיכנס לתור היא עם value גבוה יותר מאחת החבילות שיש כבר בתור, אז מוציאים מהתור את החבילה עם הvalue המינימלי ומכניסים את החבילה עם הvalue הגבוה יותר. בנוסף בעת הprocessing שולחים את החבילה עם ה value המקסימלי. כך בעצם נוכל למקסם את ערך הvalue הסופי שנקבל, הslack לא מהווה כאן פרמטר מאחר ולכל החבילות יש אותו הslack.

לכן במידה וה slack של כל החבילות שווה אך הvalue שונה עדיף להשתמש בbound delay אשר דואג להכניס לתור ולשלוח את החבילות עם הvalue הגבוהה ביותר.

c. במקרה בו גם הslack שונה וגם הvalue אז יש כמה אופציות:

1. **במקרה שהתור קטן והslack גבוה-** עדיף להשתמש ב**bound delay**, מאחר וכאשר אנו מניחים שהחבילות שהתקבלו הן עם slack גבוה והתור קטן אז אנו יודעים שנספיק לטפל בחבילות לפי שיגמר להם הslack ולכן נעבוד לפי הvalue ונשתמש באלגוריתם bound delay אשר ממקסם את ה value הסופי שנקבל מאחר ועובד על הכנסה ושליחה של החבילות עם הvalue המקסימלי.

2. **במקרה שהתור גדול והslack גבוה-** עדיף להשתמש ב **EDF**, מאחר ואם נשתמש בbound delay הסיכוי שנאבד חבילות הוא גבוהה יותר. bound delay פועל על פי הvalue המקסימלי אותו הוא מכניס לתור ושולח, ואילו אנו יודעים שהתור מספיק גדול וגם הslack של החבילות גבוה ולכן הסיכוי שיאבדו חבילות בEDF הוא יותר נמוך.

3. **במקרה שהתור קטן והslack נמוך**- עדיף להשתמש ב**EDF**, כאשר הslack נמוך הסיכוי שיאבדו חבילות בעקבות זה שלא הספקנו לטפל בהם, הוא גדול. ולכן עדיף להשתמש בEDF אשר ידוע שבעזרתו נטפל בחבילות עם ה slack הגבוה יותר, כלומר נטפל ביותר חבילות.

4. **במקרה שהתור גדול והslack נמוך**- עדיף להשתמש ב**bound delay**, במקרה זה שהתור גדול והslack נמוך אנו יודעים שבכל מקרה יהיה איבוד חבילות מאחר והslack של החבילות יגיע יחסית מהר ל0. לכן נרצה שאת החבילות עם הvalue המקסימלי נכניס לתור ונשלח, כלומר שהחבילות שיאבדו יהיו עם value מינימלי.

d. אלגוריתם ידוע לבקרת גודש – Least Slack Time(LST). אלגוריתם דינאמי.

בLST לכל המשימות במערכת יש עדיפות מסוימת בהתאם לslack שלהם, למשימה עם הslack הנמוך ביותר יש את העדיפות הגבוהה ביותר ולהיפך.
סדרי העדיפויות לכל משימה מוקצים באופן דינאמי.

הslack מחושב כך – slack = (D – t – e)
כאשר : D = הזמן לטיפול בחבילה.
t = הזמן שעבר מאז תחילת המחזור.
e = זמן שנותר לביצוע המשימה.

המשימה בעלת הslack המינימלי נשלחת ראשונה מכיוון שיש לה את העדיפות הגבוהה ביותר.
בזמן t, הslack של החבילה שווה ל(D-t) מינוס הזמן הנדרש להשלמת החלק שנותר במשימה.
השימוש הנפוץ ביותר בו הוא בembedded systems, במיוחד אלו עם מעבדים מרובים. זה מטיל את האילוץ הפשוט שלכל תהליך בכל מעבד זמין יש זמן ריצה זהה, ולתהליכים בודדים אין זיקה למעבד מסוים. זה מה שמעניק לו התאמה לembedded systems.

חסרונות –

- אלגוריתם זה מסובך, והוא דורש מידע נוסף כמו- זמני ביצוע.
- עובד רק על מצב המערכת הנוכחי.

יתרונות-

- אלגוריתם זה דינאמי ולכן יודע לייעל את הטיפול בחבילות לפי הslack העדכני של כל חבילה.
- אלגוריתם זה יכול לאפשר שימוש של עד 100% של המעבד.