I. EXAMPLES OF STABILITY CONDITIONS

Example: Consider a general arrival process and let $N(t)$ represent the number of arrivals during $[0, t]$. Suppose the following conditions hold:

- $\frac{N(t)}{t} \to \lambda$ as $t \to \infty$.
- Packet lengths are $B_1, B_2, B_3, ..., \text{(not necessarily i.i.d)}$.
- $\lim_{n \to \infty} \frac{1}{n} \sum_{i=1}^{n} B_i = \bar{B}$ (w.p.1).

**Claim:** The bit rate of this process is $\lambda \bar{B}$.

**Proof:** We have:

$$X(t) = \sum_{i=1}^{N(t)} B_i$$

Dividing by $t$ and taking limits we have:

$$\lim_{t \to \infty} \frac{X(t)}{t} = \lim_{t \to \infty} \frac{1}{t} \sum_{i=1}^{N(t)} B_i = \lim_{t \to \infty} \frac{N(t)}{t} \sum_{i=1}^{N(t)} \frac{B_i}{N(t)} = \lambda \bar{B}$$

completing the proof.

If these arrivals feed into a single server queue with time average rate $\mu$, then stability condition is $\lambda \bar{B} \leq \mu$.

II. SERVICE TIME DESCRIPTION

Queueing systems may not involve bits. In such systems, the arrival process consists of jobs which have service times. The service time $S_i$ of job $i$ represents the amount of time this job will take to complete when it gets to the server.

How does this relate to our original concept (as shown in Fig 1)?

Define an equivalent virtual system as follows:

- A single server system with processing rate $\mu = 1 \text{ bit/s}$ in which “packets” arrive exactly when the jobs arrive in the actual system.
- Packet length $B_i$ in the virtual system = (Service time $S_i$ of the job in the actual system) $\left(\frac{1 \text{ bit}}{\text{sec}}\right)$

Note unit conversion: $S_i \text{ seconds} = B_i \text{ bits}$ ($B_i = S_i \times (1 \text{ bit/sec})$).

---

![Fig. 1. A single server queue](image-url)
Define $V(t)$ as the unfinished work (in bits) in the virtual system as a function of time. This equals the sum of the residual service times of all jobs in the actual system (ex: If FIFO order, then $V(t)$ is the sum of service times of all jobs waiting in the buffer + residual service time of job in server). If $\lim_{t \to \infty} \frac{V(t)}{t} = 0$ we have rate stability.

Rate stability in the virtual system occurs if and only if the bit arrival rate $r$ is less than or equal to the time average server rate, which in this case is equivalent to:

$$\lambda \overline{B} \leq 1 \text{ bit/s}$$

Thus in this “service time” context, we have stability if and only if:

$$\lambda \overline{Z} \leq 1 \quad (1)$$

Example: Consider two streams entering a single server (Fig. 2):
- $N_1(t)$ has rate $\lambda_1$, average service time $\overline{S}_1$ seconds
- $N_2(t)$ has rate $\lambda_2$, average service time $\overline{S}_2$ seconds

Let $\overline{B}_1 = \overline{S}_1$ bits and $\overline{B}_2 = \overline{S}_1$ bits. Now, in an equivalent virtual system with constant service rate $\mu = 1$ bit/sec:
- $N_1(t)$ has packet arrival rate $\lambda_1$, average packet size $\overline{B}_1$
- $N_2(t)$ has packet arrival rate $\lambda_2$, average packet size $\overline{B}_2$

Total bit rate for virtual system is $\lambda_1 \overline{B}_1 + \lambda_2 \overline{B}_2$. Thus, the stability condition for the virtual packet system is:

$$\lambda_1 \overline{B}_1 + \lambda_2 \overline{B}_2 \leq 1$$

This stability condition implies:

$$\lim_{t \to \infty} \frac{V(t)}{t} = 0$$

In the actual system, this stability condition translates to:

$$\lambda_1 \overline{S}_1 + \lambda_2 \overline{S}_2 \leq 1 \quad (2)$$

For this example,

$$\lambda = \lambda_1 + \lambda_2$$

$$\overline{S} = \frac{\lambda_1}{\lambda_1 + \lambda_2} \overline{S}_1 + \frac{\lambda_2}{\lambda_1 + \lambda_2} \overline{S}_2$$

It can be seen that the stability condition from (1)

$$\lambda \overline{S} \leq 1$$

exactly corresponds to (2).
The delay of a packet is defined as the total amount of time the packet spends in the system before it departs. In particular, we define the following variables:

- \( a_n \triangleq \) arrival time of \( n \)th packet
- \( d_n \triangleq \) departure time of \( n \)th packet
- \( W_n \triangleq d_n - a_n = \) delay of packet \( n \)

Example: Consider a single server queue with a constant service rate \( \mu = 1 \text{ bits/s} \). \( a_1 = 1, d_1 = 3, W_1 = 2 \) \( a_2 = 2, d_2 = 4, W_2 = 2 \) (see Fig. 3)

**Definition 1:** \( W_n^{(q)} = \) total time spent by packet \( n \) in “queue” (buffer).

How can we express \( W_n \) and \( W_n^{(q)} \) in terms of the unfinished work \( U(t) \)?

In a FIFO system with constant server rate \( \mu \) bits/s,

\[
W_n = \frac{U(a_n^+)}{\mu}
\]

\[
W_n^{(q)} = \frac{U(a_n^-)}{\mu}
\]

**IV. CALCULUS FOR NETWORK DELAY**

**Definition 2:** A traffic stream \( X(t) \) is *leaky bucket constrained* with rate \( r \) (bits/sec) and burst \( \sigma \) (bits) if the total bit arrivals satisfy:

\[
X[t, t + T] \leq rT + \sigma
\]

for all time intervals starting at any time \( t \) and having any length \( T \). As shortened notation, we indicate that \( X(t) \) is such a process by saying that \( X(t) \) is _leaky bucket constrained with parameters \( (r, \sigma) \)_.

Example: Packets of length 2Kbits periodically arrive every 5 seconds. What are the associated \( (r, \sigma) \) parameters?

\[X[t, t + T] \leq \left( \frac{T}{5} + 1 \right) 2\text{Kbits}\]

For this example, the long term rate \( \lambda = \frac{2}{5} \text{ Kb/s} \) and the burst size \( \sigma = 2\text{Kb} \).

Example: Given a leaky bucket input with parameters \( (\lambda, \sigma) \), show that

\[
\lim_{t \to \infty} \frac{X(t)}{t} \leq \lambda
\]
Proof:

\[
\lim_{{t \to \infty}} \frac{X(t)}{t} = \lim_{{t \to \infty}} \frac{X[0, t]}{t} \\
\leq \lim_{{t \to \infty}} \frac{\lambda t + \sigma}{t} = \lambda
\]

Example: Suppose we are constrained to send according to leaky bucket parameters \((\lambda, \sigma) = (1Kb/s, 10Kb)\). At time \(t = 0\), we send our first packet with length 10Kb. Suppose the next packet is 2Kb. What is the earliest time \(T^*\) it can be sent?

By definition,

\[
X[0, T^*) \leq (1Kb/s) T^* + 10kB
\]

Using, \(X[0, T^*] = 12Kb\), we obtain \(T^* = 2s\).

How can we manufacture a \((\lambda, \sigma)\) process given an arbitrary input process?

Take any input \(X(t)\) and “filter” it using a simple token bucket scheme. Tokens have size 1 bit and arrive periodically every \(\frac{1}{\lambda}\) seconds. A packet of length \(B\) cannot depart if there are less than \(B\) tokens in the bucket. When it departs it takes away exactly \(B\) tokens from the bucket. If the bucket has \(B - 1\) tokens and a new token arrives, the packet can depart and the new token is consumed immediately. However, if the new token arrives when the data buffer is empty and the token bucket is full \((\sigma - 1)\), then the new token cannot be stored and so is removed and never used. In another implementation, tokens could be of size \(R(>1)\) bits so that they need not be generated as rapidly as before. In this case, \(\lambda = \frac{R}{T_{\text{token}}}\) where \(T_{\text{token}}\) is the token arrival period.