این کد یک شبیه‌ساز ساده برای ارسال و دریافت پکت‌ها در یک شبکه است. در این شبیه‌ساز، الگوریتم Reno برای مدیریت کنترل جریان و بازیابی خطا استفاده می‌شود. الگوریتم Reno یکی از الگوریتم‌های کنترل جریان TCP است که از کنترل جریان از طریق کنترل از دست رفتن پکت‌ها و زمان‌بندی مجدد پکت‌ها برای تنظیم سرعت ارسال پکت‌ها استفاده می‌کند.

حال خط به خط توضیحات این کد به شرح زیر است:

1. \*\*import socket, time, random\*\*: وارد کردن ماژول‌های مورد نیاز برای استفاده از سوکت‌ها، زمان‌دهی و ایجاد اعداد تصادفی.

2. \*\*RTT = 0.1\*\*: تعیین زمان چرخه زمانی رفت و برگشت (RTT) به عنوان 0.1 ثانیه.

3. \*\*PACKET\_LOSS\_RATE = 0.1\*\*: تعیین نرخ از دست رفتن پکت (Packet Loss Rate) به عنوان 0.1.

4. \*\*CONGESTION\_WINDOW\_SIZE = 4\*\*: تعیین اندازه پنجره تحکیم (Congestion Window) به عنوان 4.

5. \*\*packet\_in\_flight = 0\*\*: تعیین متغیر برای نگه‌داری تعداد پکت‌های در حال ارسال.

6. \*\*SERVER\_ADDRESS = ('localhost', 12345)\*\*: تعیین آدرس و پورت سرور.

7. \*\*RETRANSMISSION\_TIMEOUT = 0.5\*\*: تعیین زمان تاخیر برای ارسال مجدد پکت‌ها.

8. \*\*def send\_packet(packet\_number, sock)\*\*: تابعی برای ارسال یک پکت با شماره مشخص.

9. \*\*def wait\_for\_ack(sock)\*\*: تابعی برای انتظار دریافت ACK از سرور.

10. \*\*def timeout\_or\_packet\_loss()\*\*: تابعی برای مدیریت زمان تاخیر یا از دست رفتن پکت‌ها.

11. \*\*def retransmit\_packets(sock)\*\*: تابعی برای ارسال مجدد پکت‌ها در صورت مشکل.

12. \*\*def initialize()\*\*: تابعی برای مقداردهی اولیه و اتصال به سرور.

13. \*\*sock = initialize()\*\*: مقداردهی اولیه و اتصال به سرور.

14. \*\*for i in range(10)\*\*: حلقه برای ارسال 10 پکت به سرور.

15. \*\*send\_packet(i, sock)\*\*: ارسال هر پکت.

16. \*\*ack\_received = wait\_for\_ack(sock)\*\*: انتظار دریافت ACK از سرور.

17. \*\*if packet\_in\_flight > CONGESTION\_WINDOW\_SIZE or not ack\_received\*\*: بررسی برای تأخیر یا از دست رفتن پکت‌ها.

18. \*\*timeout\_or\_packet\_loss()\*\*: مدیریت زمان تاخیر یا از دست رفتن پکت‌ها.

19. \*\*retransmit\_packets(sock)\*\*: ارسال مجدد پکت‌ها در صورت مشکل.

20. \*\*sock.close()\*\*: بستن سوکت پس از اتمام کار.

این دو تابع در این کد برای ارسال پکت‌ها و دریافت ACK از سرور استفاده می‌شوند. بیایید آن‌ها را خط به خط توضیح دهیم:

1. \*\*def send\_packet(packet\_number, sock):\*\*

- این تابع برای ارسال یک پکت به سرور استفاده می‌شود.

- `packet\_number`: شماره پکتی که قرار است ارسال شود.

- `sock`: شیء سوکت برای ارتباط با سرور.

- `global packet\_in\_flight`: این دستور اعلام می‌کند که متغیر `packet\_in\_flight` که در بخش بیرونی تعریف شده است، در داخل این تابع نیز استفاده می‌شود.

- ابتدا پیامی در خروجی چاپ می‌شود که اعلام می‌کند چه پکتی در حال ارسال است.

- با استفاده از `random.random()`، یک عدد تصادفی بین 0 و 1 تولید می‌شود و با مقایسه با `PACKET\_LOSS\_RATE` که مشخص کننده نرخ از دست رفتن پکت‌ها است، تصمیم می‌گیرد که آیا پکت ارسال شود یا نه.

- اگر عدد تصادفی کمتر از `PACKET\_LOSS\_RATE` باشد، یعنی پکت از دست رفته و پیام "Packet Lost!" چاپ می‌شود.

- در غیر این صورت، پکت ارسال می‌شود، روی آن انتظار می‌رود که ACK دریافت شود، و متغیر `packet\_in\_flight` افزایش می‌یابد.

- در صورت بروز خطا در ارسال پکت، پیام خطا چاپ می‌شود.

2. \*\*def wait\_for\_ack(sock):\*\*

- این تابع برای دریافت پاسخ ACK از سرور استفاده می‌شود.

- `sock`: شیء سوکت برای ارتباط با سرور.

- ابتدا سعی می‌شود که اطلاعات از سرور دریافت شود.

- اگر داده‌ای دریافت شود، آن را با استفاده از `decode()` به رشته تبدیل کرده و پیام "Received ACK: " همراه با محتوای دریافتی را چاپ می‌کند.

- در صورتی که timeout رخ دهد، یعنی زمان مقرر برای دریافت ACK از سمت سرور گذشته و پاسخی دریافت نشد، تابع `False` را برمی‌گرداند.

الگوریتم کنترل ازدحام TCP Reno یکی از الگوریتم‌های کنترل جریان در پروتکل TCP است که برای مدیریت انتقال داده‌ها در شبکه و کنترل ازدحام (Congestion Control) استفاده می‌شود. این الگوریتم از دو مکانیزم اصلی به نام‌های Slow Start و Fast Recovery برای کنترل جریان و بازیابی پس از از دست رفتن پکت‌ها استفاده می‌کند.

الگوریتم کنترل ازدحام TCP Reno به شرح زیر عمل می‌کند:

1. \*\*Slow Start\*\*:

- در ابتدا، اندازه پنجره تحکیم (Congestion Window) به طور پیش‌فرض یک پکت است.

- هنگامی که یک پکت از سمت مقصد ACK می‌شود، اندازه پنجره تحکیم دو برابر می‌شود.

- این فرآیند ادامه می‌یابد تا زمانی که محدودیت‌های ازدحام رخ دهند.

2. \*\*Congestion Avoidance\*\*:

- بعد از رسیدن به حداکثر اندازه مجاز پنجره تحکیم، الگوریتم از حالت Slow Start به حالت Congestion Avoidance وارد می‌شود.

- در این حالت، اندازه پنجره تحکیم به طور خطی افزایش می‌یابد تا زمانی که محدودیت‌های ازدحام رخ دهند.

3. \*\*Fast Recovery\*\*:

- هنگامی که یک پکت از دست می‌رود (تاخیر از زمان مقرر برای دریافت ACK گذشته و ACK دریافت نشده است)، پنجره تحکیم به نصف کاهش می‌یابد و عملکرد شبکه به حالت Fast Recovery وارد می‌شود.

- در این حالت، پس از دریافت ACK برای پکت‌های بعدی، پنجره تحکیم به طور خطی افزایش می‌یابد.

- هنگامی که دوباره یک پکت از دست داده می‌شود، الگوریتم به حالت Slow Start باز می‌گردد.

به طور کلی، الگوریتم کنترل ازدحام TCP Reno با استفاده از این مکانیزم‌ها، تلاش می‌کند تا با تنظیم پنجره انتقال داده‌ها و بازیابی سریع از از دست رفتن پکت‌ها، عملکرد بهینه شبکه TCP را ارتقا دهد و از ترافیک زیاد و ازدحام شبکه جلوگیری کند.