به نام خدا

تمرین کامپیوتری چهارم

استاد : دکتر یزدانی

اعضای گروه :

علی دارابی – 810100264

حسام رمضانیان – 810100248

شرح تقسیم کار :

تمام بخش ها بصورت دو نفره و بر روی یک سیستم انجام شده است.

قسمت اول : پیاده‌سازی یک سرور و کلاینت ساده TCP

نمایی از کار کردن بخش سرور :

همانطور که مشاهده می‌کنید در ابتدا سه کلاینت با سرور ارتباط برقرار می‌کنند و three-way handshake انجام می‌شود و سپس هر کدام برای سرور پیام Hello را می‌فرستند که سرور این پیام ها را به درستی دریافت کرده است. همچنین آیدی کلاینتی که پیام را فرستاده است در پیام مشخص است. توجه داشته باشید که، آیدی که به عنوان آیدی کلاینت نمایش داده می‌شود، شماره File Descriptor مربوط به سوکت است به همین علت از 4 شروع شده و در صورت بسته شدن سوکت مجددا می‌توان از همان File Descriptor برای سوکت های بعدی استفاده کرد.

```
Server: waiting for connections...
connect to client : 4
Server: connection accepted
Server: received 1,SYN
Server: sent SYN_ACK
Server: received 1,ACK
Server: sent Hello
Packet ID: 1
Packet Data: Hello
connect to client : 5
Server: connection accepted
Server: received 2,SYN
Server: sent SYN_ACK
Server: received 2,ACK
Server: sent Hello
Packet ID: 2
Packet Data: Hello
connect to client : 4
Server: connection accepted
Server: received 3,SYN
Server: sent SYN_ACK
Server: received 3,ACK
```

```
Server: sent Hello
Packet ID: 3
Packet Data: Hello
```

نمایی از کار کردن بخش کلاینت :

همانطور که مشاهده می‌شود سه کلاینت پیام ها را به درستی ارسال و دریافت کرده اند. (چون همه کلاینت ها پیام را به سرور می فرستند، تمام Packet ID ها صفر است، زیرا که آیدی سرور 0 در نظر گرفته شده.)

```
hesamhrf@hesamhrf-OMEN-by-HP-Laptop-15-dh1xxx:~/CN$ ./my_program
c
Client: sent SYN
Client: received 0,SYN_ACK
Client: sent ACK
Client: sent Hello
Client: received 0,Hello
Packet ID: 0
Packet Data: Hello
Client: sent SYN
Client: received 0,SYN_ACK
Client: sent ACK
Client: sent Hello
Client: received 0,Hello
Packet ID: 0
Packet Data: Hello
Client: sent SYN
Client: received 0,SYN_ACK
Client: sent ACK
Client: sent Hello
Client: received 0,Hello
Packet ID: 0
Packet Data: Hello
```

قسمت دوم : پیاده‌سازی یک پروتکل کنترل ازدحام

نحوه کارکرد این الگوریتم :

پنجره ازدحام: از پنجره‌ای به نام `cwnd` استفاده می‌کند تا میزان داده‌ای که می‌تواند بدون دریافت تاییدیه (ACK) ارسال کند را کنترل کند.

آغاز آهسته (Slow Start): در ابتدا، `cwnd` با افزایش نمایی رشد می‌کند تا زمانی که به آستانه‌ای به نام `ssthresh` برسد. این آستانه، آستانه‌ی آغاز آهسته نام دارد و اندازه پنجره ازدحام را تا جایی که شبکه بتواند آن را پشتیبانی کند، افزایش می‌دهد.

اجتناب از ازدحام (Congestion Avoidance): وقتی `cwnd` به `ssthresh` رسید، الگوریتم به حالت اجتناب از ازدحام سوئیچ می‌کند. در این حالت، `cwnd` به صورت خطی افزایش می‌یابد تا از ایجاد ازدحام در شبکه جلوگیری کند.

ارسال مجدد سریع (Fast Retransmit): وقتی یک بسته گم می‌شود، TCP New Reno به سرعت آن را پس از دریافت سه تاییدیه‌ی (ACK) تکراری برای همان بسته، ارسال مجدد می‌کند، بدون اینکه منتظر تایم اوت شود. این حالت به عنوان "ارسال مجدد سریع" شناخته می‌شود.

بازیابی سریع (Fast Recovery): پس از ارسال مجدد بسته‌ی گم شده، New Reno وارد بازیابی سریع می‌شود. در این مرحله:

- اندازه `ssthresh` را به نصف اندازه فعلی `cwnd` کاهش می‌دهد تا از ازدحام بیشتر جلوگیری کند.
- `cwnd` به اندازه `(ssthresh + 3)` کاهش می‌یابد و سپس به تدریج به صورت افزایشی افزایش می‌یابد تا شبکه دوباره به حالت عادی برگردد.

- پس از اتمام بازیابی سریع، الگوریتم دوباره به حالت اجتناب از ازدحام برمی‌گردد. cwnd برابر با ssthresh می‌شود.

نمایی از کار کردن بخش سرور :

در این شبیه‌سازی ما فرض کردیم پکت های 10 و 23 loss شده اند به همین دلیل سرور ACK پکت قبل از این پکت ها را برای کلاینت تکرار می‌کند.

```
make && ./my_program
s
Server: waiting for connections...
connect to client : 4
Server: connection accepted
Server: received 0,SYN
pack id : 0
Server: sent SYN_ACK
Server: received 1,ACK
Server: received 2,Hello
pack id : 2
Server: sent ACKof packet (2)
Server: received 3,Hello
pack id : 3
Server: sent ACKof packet (3)
Server: received 4,Hello
pack id : 4
Server: sent ACKof packet (4)
Server: received 5,Hello
pack id : 5
Server: sent ACKof packet (5)
Server: received 6,Hello
pack id : 6
Server: sent ACKof packet (6)
Server: received 7,Hello
pack id : 7
```

```
Server: sent ACKof packet (7)
Server: received 8,Hello
pack id : 8
Server: sent ACKof packet (8)
Server: received 9,Hello
pack id : 9
Server: sent ACKof packet (9)
Server: received 10,Hello
pack id : 9
Server: sent ACKof packet (9)
Server: received 11,Hello
pack id : 9
Server: sent ACKof packet (9)
Server: received 12,Hello
pack id : 9
Server: sent ACKof packet (9)
Server: received 10,Hello
pack id : 10
Server: sent ACKof packet (10)
Server: received 11,Hello
pack id : 11
Server: sent ACKof packet (11)
Server: received 12,Hello
pack id : 12
Server: sent ACKof packet (12)
Server: received 13,Hello
pack id : 13
Server: sent ACKof packet (13)
Server: received 14,Hello
pack id : 14
Server: sent ACKof packet (14)
Server: received 15,Hello
pack id : 15
Server: sent ACKof packet (15)
Server: received 16,Hello
pack id : 16
Server: sent ACKof packet (16)
Server: received 17,Hello
pack id : 17
```

```
Server: sent ACKof packet (17)
Server: received 18,Hello
pack id : 18
Server: sent ACKof packet (18)
Server: received 19,Hello
pack id : 19
Server: sent ACKof packet (19)
Server: received 20,Hello
pack id : 20
Server: sent ACKof packet (20)
Server: received 21,Hello
pack id : 21
Server: sent ACKof packet (21)
Server: received 22,Hello
pack id : 22
Server: sent ACKof packet (22)
Server: received 23,Hello
pack id : 22
Server: sent ACKof packet (22)
Server: received 24,Hello
pack id : 22
Server: sent ACKof packet (22)
Server: received 25,Hello
pack id : 22
Server: sent ACKof packet (22)
Server: received 23,Hello
pack id : 23
Server: sent ACKof packet (23)
Server: received 24,Hello
pack id : 24
Server: sent ACKof packet (24)
Server: received 25,Hello
pack id : 25
Server: sent ACKof packet (25)
Server: received 26,Hello
pack id : 26
Server: sent ACKof packet (26)
Server: received 27,Hello
pack id : 27
```

```
Server: sent ACKof packet (27)
Server: received 28,Hello
pack id : 28
Server: sent ACKof packet (28)
Server: received 29,Hello
pack id : 29
Server: sent ACKof packet (29)
Server: received 30,Hello
pack id : 30
Server: sent ACKof packet (30)
Server: received 31,Hello
pack id : 31
Server: sent ACKof packet (31)
Server: received 32,Hello
pack id : 32
Server: sent ACKof packet (32)
Server: received 33,Hello
pack id : 33
Server: sent ACKof packet (33)
Server: received 34,Hello
pack id : 34
Server: sent ACKof packet (34)
Server: received 35,Hello
pack id : 35
Server: sent ACKof packet (35)
```

نمایی از کار کردن بخش کلاینت :

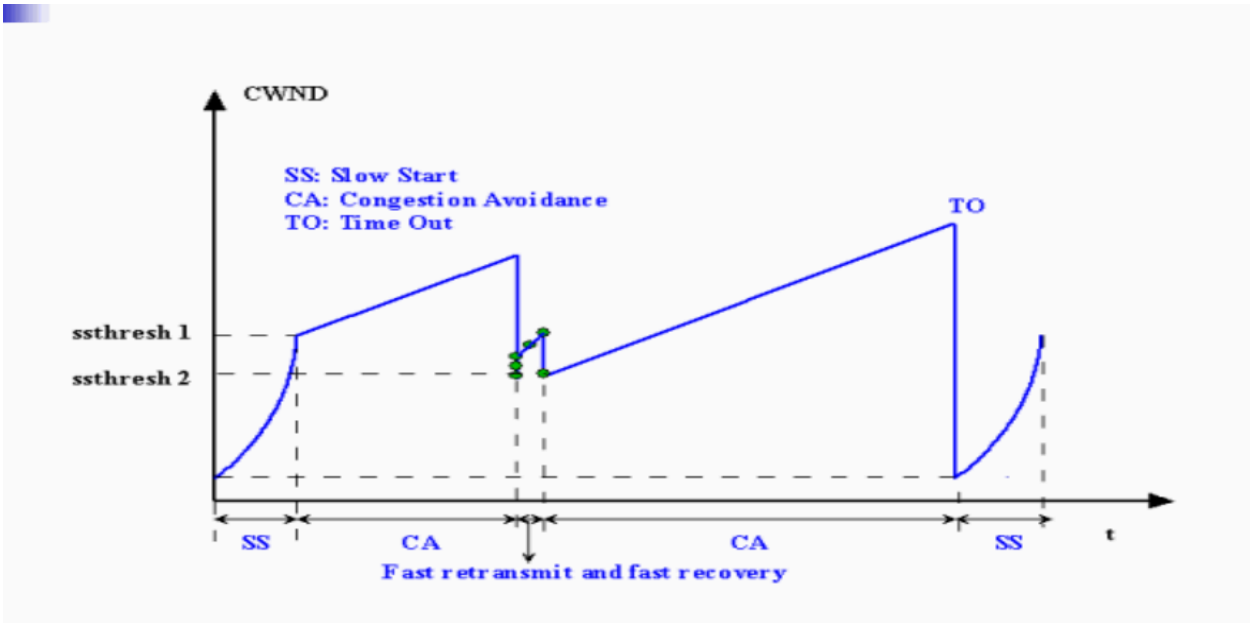
همانطور که مشاهده می‌شود کلاینت وقتی متوجه می‌شود که سه ACK تکراری برای پکت 9 دریافت کرده است، پکت 10 را Fast Retransmit می‌کند و وارد فاز Fast Recovery می‌شود. همچنین زمانی که متوجه می‌شود که سه ACK تکراری برای پکت 22 دریافت، پکت 23 را Fast Retransmit می‌کند و وارد فاز Fast Recovery می‌شود.

```
 Client: sent SYN
Client: received 0,SYN_ACK
Client: sent ACK
Client: sent Hello(2) with cwnd = 1
Client: received 2,ACK
Client: sent Hello(3) with cwnd = 2
Client: received 3,ACK
Client: sent Hello(4) with cwnd = 4
Client: received 4,ACK
Client: sent Hello(5) with cwnd = 8
Client: received 5,ACK
Client: sent Hello(6) with cwnd = 16
Client: received 6,ACK
Client: sent Hello(7) with cwnd = 32
Client: received 7,ACK
Entering Congestion Avoidance
Client: sent Hello(8) with cwnd = 64
Client: received 8,ACK
Client: sent Hello(9) with cwnd = 65
Client: received 9,ACK
Client: sent Hello(10) with cwnd = 66
Client: received 9,ACK
Client: sent Hello(11) with cwnd = 66
Client: received 9,ACK
Client: sent Hello(12) with cwnd = 66
Client: received 9,ACK
Entering Fast Recovery
retransmit packet 10
Client: sent Hello(10) with cwnd = 36
```

```
Client: received 10,ACK
Client: sent Hello(11) with cwnd = 37
Client: received 11,ACK
Client: sent Hello(12) with cwnd = 38
Client: received 12,ACK
Entering Congestion Avoidance
Client: sent Hello(13) with cwnd = 33
Client: received 13,ACK
Client: sent Hello(14) with cwnd = 34
Client: received 14,ACK
Client: sent Hello(15) with cwnd = 35
Client: received 15,ACK
Client: sent Hello(16) with cwnd = 36
Client: received 16,ACK
Client: sent Hello(17) with cwnd = 37
Client: received 17,ACK
Client: sent Hello(18) with cwnd = 38
Client: received 18,ACK
Client: sent Hello(19) with cwnd = 39
Client: received 19,ACK
Client: sent Hello(20) with cwnd = 40
Client: received 20,ACK
Client: sent Hello(21) with cwnd = 41
Client: received 21,ACK
Client: sent Hello(22) with cwnd = 42
Client: received 22,ACK
Client: sent Hello(23) with cwnd = 43
Client: received 22,ACK
Client: sent Hello(24) with cwnd = 43
Client: received 22,ACK
Client: sent Hello(25) with cwnd = 43
Client: received 22,ACK
Entering Fast Recovery
retransmit packet 23
Client: sent Hello(23) with cwnd = 24
Client: received 23,ACK
Client: sent Hello(24) with cwnd = 25
Client: received 24,ACK
Client: sent Hello(25) with cwnd = 26
```

```
Client: received 25,ACK
Entering Congestion Avoidance
Client: sent Hello(26) with cwnd = 21
Client: received 26,ACK
Client: sent Hello(27) with cwnd = 22
Client: received 27,ACK
Client: sent Hello(28) with cwnd = 23
Client: received 28,ACK
Client: sent Hello(29) with cwnd = 24
Client: received 29,ACK
Client: sent Hello(30) with cwnd = 25
Client: received 30,ACK
Client: sent Hello(31) with cwnd = 26
Client: received 31,ACK
Client: sent Hello(32) with cwnd = 27
Client: received 32,ACK
Client: sent Hello(33) with cwnd = 28
Client: received 33,ACK
Client: sent Hello(34) with cwnd = 29
Client: received 34,ACK
Client: sent Hello(35) with cwnd = 30
Client: received 35,ACK
```

تصویری از نحوه کار این پروتکل :



قسمت سوم: پیاده‌سازی یک پروتکل کنترل جریان

توضیح کارکرد پروتکل Go-Back-N :

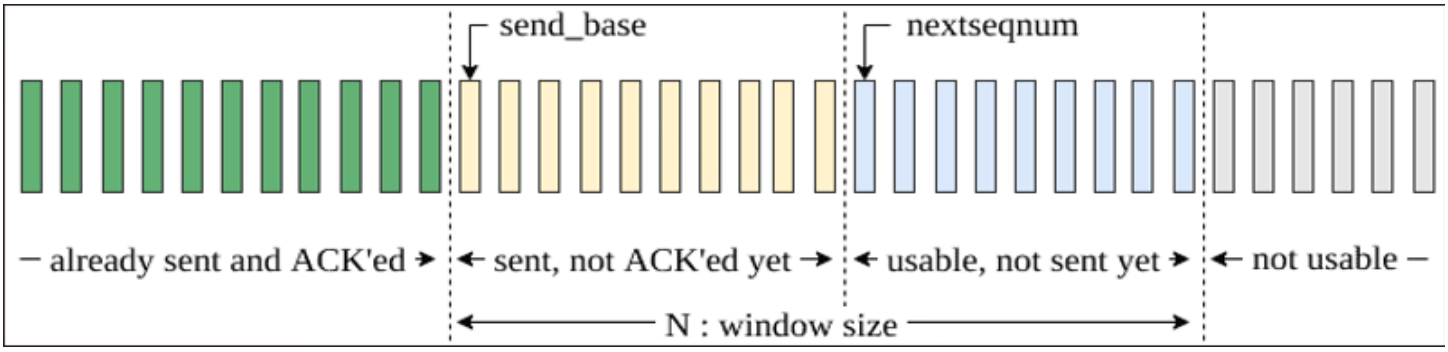
در Go-Back-N، فرستنده جریان بسته ها را کنترل می کند، به این معنی که ما یک گیرنده ساده داریم. بنابراین، ابتدا با بحث درباره نحوه مدیریت بسته های داده توسط فرستنده شروع می کنیم.

فرستنده یک سری فریم برای ارسال دارد. ما اندازه پنجره را N فرض می کنیم. علاوه بر این، دو اشاره‌گر send_base و nextseqnum وجود دارد. برای شروع، فرستنده با ارسال اولین فریم کار خود را آغاز می‌کند. در ابتدا، مقدار send_base برابر با ۰ و مقدار nextseqnum نیز برابر با ۰ است. تا زمانی که بسته‌های بیشتری برای ارسال وجود دارند و مقدار nextseqnum کوچکتر از send_base + N است؛ فرستنده بسته‌ای که به وسیله اشاره‌گر nextseqnum نشان داده می‌شود را ارسال کرده و سپس nextseqnum را افزایش می‌دهد. در همین حال، مقدار send_base پس از دریافت بسته‌های تایید (ACK) از گیرنده، افزایش می‌یابد. دریافت پیام‌های تاییدیه تکراری هیچ سازوکاری را فعال نمی‌کند.

پیاده‌سازی گیرنده در پروتکل Go-Back-N تا حد ممکن ساده است، گیرنده فقط شماره sequence مورد انتظاری که قرار است بعدی دریافت شود را دنبال می‌کند: nextseqnum. هیچ بافری در گیرنده وجود ندارد؛ بسته‌هایی که خارج از ترتیب دریافت می‌شوند، به سادگی دور انداخته می‌شوند. به طور مشابه، بسته‌های خراب نیز بدون هیچگونه اعلان یا پیام خطا دور انداخته می‌شوند. گیرنده همیشه پس از دریافت یک بسته جدید (موفقیت‌آمیز باشد یا خیر)، تاییدیه آخرین بسته‌ای که به ترتیب درست

دریافت شده را ارسال می‌کند. بنابراین، در صورتی که مشکلی رخ دهد، پیام‌های تاییدیه تکراری تولید خواهد شد.

تصویری از نحوه کار کردن این پروتکل :



توضیح کارکرد پروتکل Selective Repeat :

پروتکل تکرار انتخابی (SRP) یک پروتکل انتقال داده قابل اعتماد در شبکه‌های کامپیوتری است که برای اطمینان از تحویل صحیح و به ترتیب بسته‌ها استفاده می‌شود. برخلاف پروتکل‌های ساده‌تر، پروتکل تکرار انتخابی با ارسال مجدد تنها بسته‌های اشتباه یا گمشده، به جای کل توالی بسته‌ها، به بهبود استفاده از پهنای باند و کاهش سربار ناشی از ارسال مجدد کمک می‌کند.

سمت فرستنده : ارسال: فرستنده یک پنجره از بسته‌ها را نگه می‌دارد که می‌تواند بدون انتظار برای تاییدیه‌ها ارسال کند. بسته‌های داخل این پنجره به ترتیب ارسال می‌شوند. بافر کردن: بسته‌های ارسال شده تا زمان دریافت تاییدیه بافر می‌شوند تا در صورت بروز خطا یا گم شدن بسته‌ها، امکان ارسال مجدد آنها فراهم شود. ارسال مجدد : اگر تاییدیه برای یک بسته خاص در مدت زمان مشخصی دریافت نشود (به دلیل از دست رفتن بسته یا خطا)، فرستنده فقط بسته خاصی که تاییدیه نشده است را دوباره ارسال می‌کند و کل توالی بسته‌ها را ارسال نمی‌کند.

سمت گیرنده : دریافت بسته‌ها: گیرنده دارای پنجره‌ای برای پذیرش بسته‌ها است. وقتی یک بسته می‌رسد، بررسی می‌کند که آیا داخل این پنجره قرار دارد یا خیر. تحویل به ترتیب: اگر بسته دریافتی بسته بعدی مورد انتظار باشد، آن را قبول کرده و تاییدیه ارسال می‌کند. اگر بسته خارج از ترتیب باشد اما همچنان در پنجره باشد، آن را بافر می‌کند. تاییدیه: گیرنده برای هر بسته‌ای که به درستی دریافت شده است، یک تاییدیه ارسال می‌کند. اگر یک بسته به ترتیب دریافت شود، پنجره را به بسته بعدی منتقل کرده و ممکن است هر بسته‌ای را که بافر کرده است و اکنون به ترتیب هستند، آزاد کند.

مدیریت خطاها: بسته‌های خارج از ترتیب: بسته‌هایی که خارج از ترتیب دریافت شده‌اند اما داخل پنجره هستند، در یک بافر ذخیره می‌شوند تا زمانی که بسته‌های گمشده قبلی برسند. تاییدیه‌های تکراری: گیرنده برای هر بسته‌ای که به درستی دریافت شده است، از جمله برای بسته‌هایی که قبلاً تأیید شده‌اند، تاییدیه ارسال می‌کند. این به فرستنده کمک می‌کند تا بسته‌هایی که نیاز به ارسال مجدد دارند را شناسایی کند.

مثال : وضعیت اولیه: پنجره فرستنده: [۰، ۱، ۲، ۳]، پنجره گیرنده: [۰، ۱، ۲، ۳] ارسال بسته‌ها: فرستنده بسته‌های ۰، ۱، ۲، ۳ را ارسال می‌کند. دریافت بسته‌ها: گیرنده بسته‌های ۰، ۱ و ۳ را به درستی دریافت می‌کند. بسته ۲ در انتقال گم شده است. گیرنده تأییدیه‌های بسته‌های ۰، ۱ و ۳ را ارسال می‌کند. مدیریت از دست دادن بسته: از آنجایی که بسته ۲ گم شده است، گیرنده بسته ۳ را ذخیره کرده و منتظر دریافت بسته ۲ می‌ماند. فرستنده، پس از دریافت تاییدیه‌ها، متوجه می‌شود که بسته ۲ تایید نشده و آن را دوباره ارسال می‌کند. تکمیل توالی: پس از دریافت و تایید بسته ۲ توسط گیرنده، می‌تواند بسته‌های ۲ و ۳ را به ترتیب به برنامه تحویل دهد. پنجره به جلو می‌لغزد تا بسته‌های بعدی را بپذیرد. ادامه ارسال: این فرآیند تکرار می‌شود و فرستنده پنجره خود را به جلو می‌لغزاند و بسته‌های جدید را ارسال می‌کند، در حالی که گیرنده همچنان بسته‌ها را به صورت لازم قبول کرده و بافر می‌کند.

قسمتی از کارکرد Host A:

همانطور که در دستور کار گفته شده بود، در ابتدا یک فایل 1.5 مگابایتی ساختیم و سپس آن را به 1000 پکت 1.5 کیلوبایتی تقسیم کردیم و شروع به ارسال این پکت ها از Host A به Host B کردیم، همانطور که در خروجی مشاهده می‌کنید بسته ها به درستی از Host A ارسال شده‌اند، چون که تعداد پکت ها خیلی زیاد است تنها ارسال 10 پکت آخر نشان داده شده است.

```
HostA sent packet 990
ACK 990
HostA sent packet 991
ACK 991
HostA sent packet 992
ACK 992
HostA sent packet 993
ACK 993
HostA sent packet 994
ACK 994
HostA sent packet 995
ACK 995
HostA sent packet 996
ACK 996
HostA sent packet 997
ACK 997
HostA sent packet 998
ACK 998
HostA sent packet 999
ACK 999
```

قسمتی از کارکرد Host B:

همانطور که مشاهده می‌کنید Host B پکت ها را به درستی دریافت کرده است و تاییدیه (ACK) آنها را برای Host A ارسال کرده است، همانطور که گفتیم چون که تعداد پکت ها خیلی زیاد است، فقط خروجی پکت 990 تا 999 را نمایش داده‌ایم و برای اطمینان از اینکه Host B پکت ها را به درستی دریافت کرده است، پکت آخر را پرینت کرده‌ایم.

```
990,ack
991,ack
992,ack
993,ack
994,ack
995,ack
996,ack
997,ack
998,ack
HostB received from Node 2 seqnum 999 :   is a line of readable text
that will be repeated to fill the file.

This is a line of readable text that will be repeated to fill the
file.

This is a line of readable text that will be repeated to fill the
file.

This is a line of readable text that will be repeated to fill the
file.

This is a line of readable text that will be repeated to fill the
file.

This is a line of readable text that will be repeated to fill the
file.

This is a line of readable text that will be repeated to fill the
file.
```

```
This is a line of readable text that will be repeated to fill the
file.

This is a line of readable text that will be repeated to fill the
file.

This is a line of readable text that will be repeated to fill the
file.

This is a line of readable text that will be repeated to fill the
file.

This is a line of readable text that will be repeated to fill the
file.

This is a line of readable text that will be repeated to fill the
file.

This is a line of readable text that will be repeated to fill the
file.

This is a line of readable text that will be repeated to fill the
file.

This is a line of readable text that will be repeated to fill the
file.

This is a line of readable text that will be repeated to fill the
file.

This is a line of readable text that will be repeated to fill the
file.

This is a line of readable text that will be repeated to fill the
file.

This is a line of readable text that will be repeated to fill the
file.

This is a line of readable text that will be repeated to fill the
file.
```

```
This is a line of readable text that will be repeated to fill the
file.

999,ack
```