UDP SendFile

•••

Protocol, Implementation & Testing Overview

Team BKR

Limitations of Original SendFile App

Data Size

- Limited to very “small” files
- Uncertain processing for “large” files
- Receiver does not know anything about file sent except name

Metadata & Sync.

- Metadata limited to file name
- Metadata transmission order uncertain
- Synchronization not clearly defined for sender and receiver

Data Integrity

- Assumes file is sent completely and in original ordering
- No mechanism to verify or prevent possible data corruption

Challenges & Targets

Communication

Establish Feedback Mechanism

Sender and receiver should be able to communicate in a simple, deterministic manner during each stage of transmission

Synchronization

Work Together & Independently

With good communication, both sender and receiver can be aware of what the other is currently doing and react appropriately

Integrity

Corrupted Data is No Data

Data transmission is ideally lossless, even minute data corruption is considered a failure condition but integrity checking should be efficient

SendFile Protocol

3 Building Blocks

1. **Syntax** - What structures shall packets assume?
2. **Semantics** - Which efforts are made to minimize error generation?
3. **Timing** - What happens if packets arrive in order other than this in which they were sent?

UDP SendFile Protocol

Revision 3: made on 10/23/2020.

Summary

The purpose of the UDP SendFile Protocol is to define policies of a transfer of a single file between sender and receiver machines. The protocol enforces feedback policy, in which the receiver notifies the sender whether the data has been correctly received or not. If an error occurs, the sender might repeatedly make file transmission attempts until the receiver recognizes no errors, or until the time cap defined on the sender is terminated.

Acknowlegements

Great thanks to members of team BKR for tailoring the foundations of the protocol during the 10/04/2020, 10/11/2020, and 10/18/2020 group meetings!

Protocol Overview

The UDP SendFile Protocol employs UDP (User Datagram Protocol) to transfer one file's data and metadata between two machines. As the UDP protocol lacks a mechanism of feedback from the receiver to the sender, the SendFile protocol elaborates upon the UDP protocol.

Two-socket connection is established between the sender and the receiver systems, which is when the sender begins transmitting data to the receiver via UDP packets.

Packet types and description

Through the UDP SendFile Protocol, the sender creates and sends UDP packets of the following kinds:

1 -- filename

2 -- total size of file in bytes (in base 10)

3 -- checksum value

4 -- file content

The file content is sent in a sequence of packets, each containing 1024 bytes of payload. If the total size in bytes of the file is 1024 bytes or less, exactly one packet carrying the file content is transmitted by the sender.

Each of the above packet types has the following structure, respectively:

```
1 byte  up to 1024 bytes
 --------------------
|  1  |  <filename>  |    ----> the filename packet
 --------------------

1 byte  up to 16 bytes
 ----------------
|  2  |  <size>  |        ----> the size packet
 ----------------

1 byte    64 bytes
 --------------------
|  3  |  <checksum>  |    ----> the checksum value packet
 --------------------

1 byte         13 bytes                1 byte          up to 1024 bytes
 ------------------------------------------------------------------------
|  4  |  <packet sequence No>  |  <More fragments?>  |  <file content>  |   ----> the file content packet
 ------------------------------------------------------------------------

1 byte     1 byte
 --------------------
|  5  |  <feedback>  |    ----> the feedback packet
 --------------------

1 byte     13 bytes
 -------------------
|  5  |  <seq_num>  |     ----> Feedback indicating a specific file content packet arrived
 -------------------
```

Syntax

- The algorithm used to produce the checksum value is Secure Hash Algorithm 2 (SHA-2) (256-bit), placed in the `<checksum>` field of the checksum packet as a 64-byte hexadecimal value.

- If any of the file content packets has not been received, or if the sent and computed checksum values differ, the receiver transmits feedback indicating that an error occured, thereby terminating the transfer. In the version that supports packet re-transfer, if the sender sees negative acknowledgement/feedback from the receiver, it re-transfers the missing packets based on the sequential number attached within the feedback packet.

- (Concern about the lack of feedback between the machines) The sender-receiver communication according to the SendFile protocol involves transmitting feedback from the receiver to the sender, which lets the sender know if a certain packet has arrived and if the entire file transfer process was carried successfully after calculating and comparing the checksum value.

Semantics

```
1 byte         13 bytes                    1 byte          up to 1024 bytes
 ---------------------------------------------------------------------------
|  4  |  <packet sequence No>  |  <More fragments?>  |  <file content>  |
 ---------------------------------------------------------------------------
```

- In the file content packet, the packet sequence number is a 13 byte field representing a base-10 integer (padded with 0s, if needed.) The reason for choosing 13 bytes is that 2^(50) / 1024 = 2^(40) = 1,099,511,627,776, the maximum possible sequence number, is a 13-digit number. For example, the 1st file content packet will have the sequence number `0000000000001`, and the 115th packet will have the sequence number `0000000000115`.

- The `<More fragments?>` field of the file content packet is either 0 (if no further fragments are expected to be transmitted by the sender) or 1 otherwise. If the file content is shorter than 1024 bytes, the `<More fragments?>` field contains 0.

Timing

- As the file content packets arrive, the receiver might implement one of two techniques. The first: it might store the data as a hash-table with the key being the sequential number of the file content packet and the value being the payload. Note that, the larger the file's content is in bytes, the larger the growing buffer on the receiver's side would be. This measure is used to eliminate the issue of terminating the transfer due to receiving file content packets in the wrong order. After all the file content packets are received, the receiver copies the payload in the correct sequential order onto an empty file it creates (with the filename indicated by the sender). The second: the receiver keeps a small buffer sufficient for storing only one file content packet's payload. As packets arrive, the receiver uses both the knowledge about the packet's payload size (1024 bytes) and the sequence number of the packet to copy the newly arrived payload into the proper place in the file, without clogging its memory.
- A short sleep period between sending packets is introducted both on the sender's and receiver's sides to increase the rates of packet arrival and processing.

Timing

Implementations

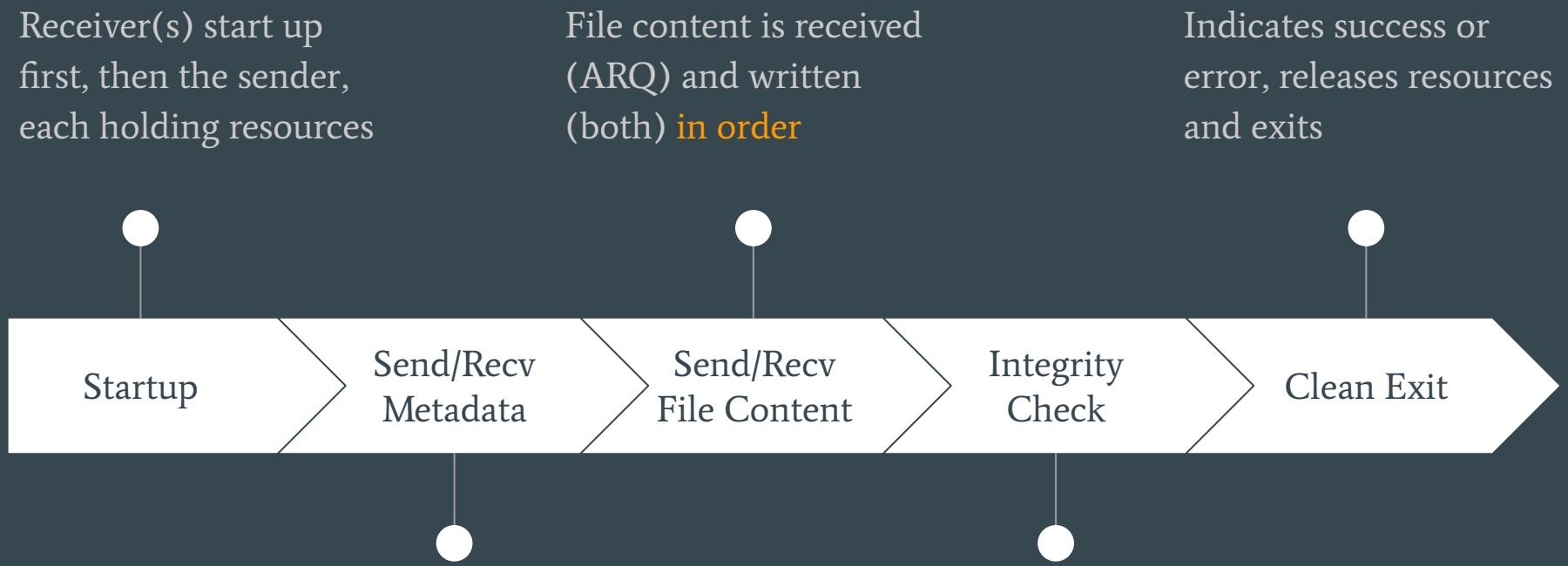
- Fail-Fast version: sender and receiver communicate only during metadata transmission and integrity check, prefers to time out whenever feedback is not received within time limit
- ARQ version: sender and receiver communicate during all major steps (including content transmission) and operate in nearly lock-step fashion
- Both versions support buffered or incremental file writing & make use of timers or packet delays

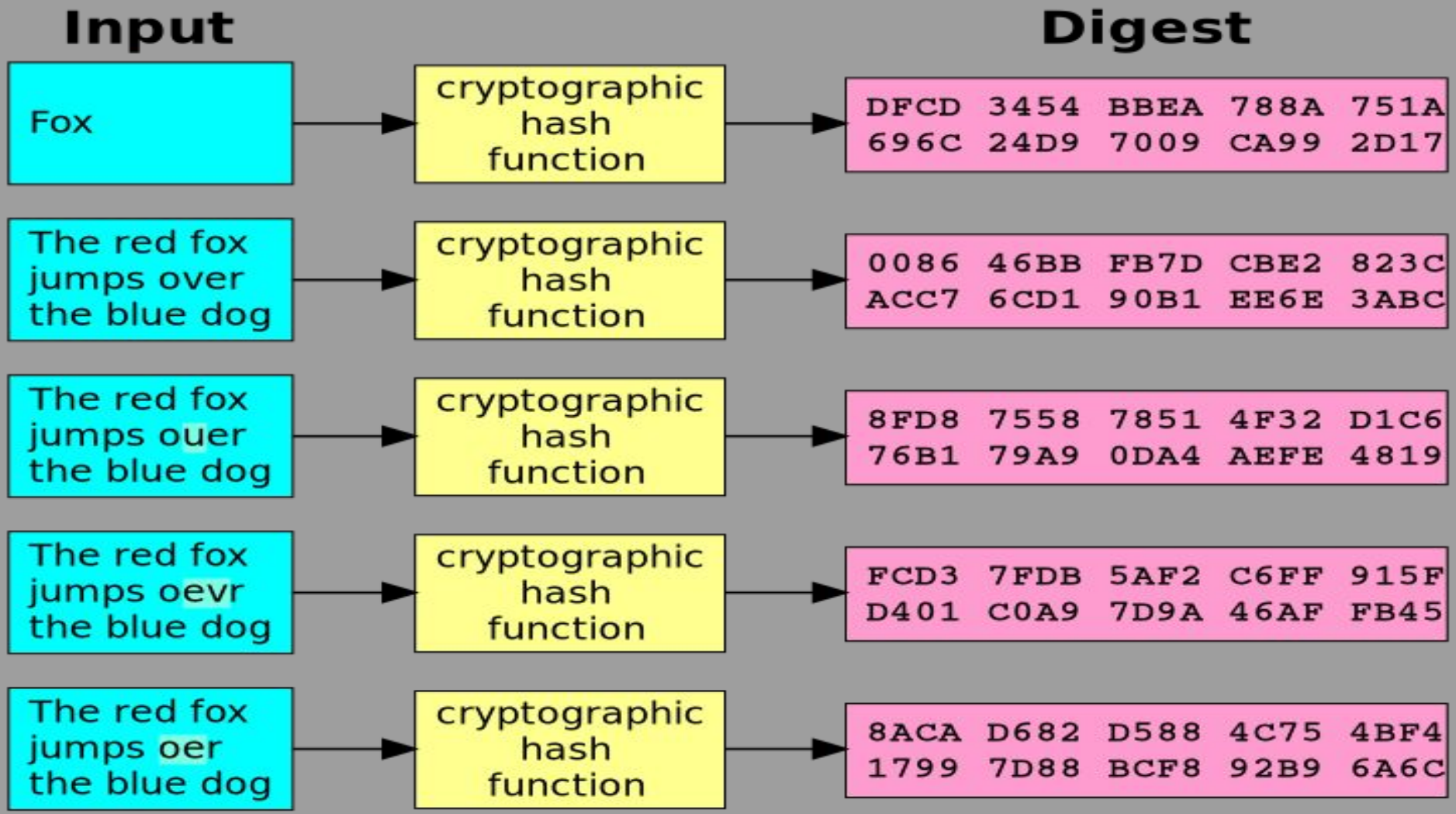
Fail-Fast Design

- Fail Fast, Fail Early, Fail Safe
- Fail at the first encounter with serious trouble
- Try -> Fail -> Adjust -> Try again...
- Adjustment is left up to user

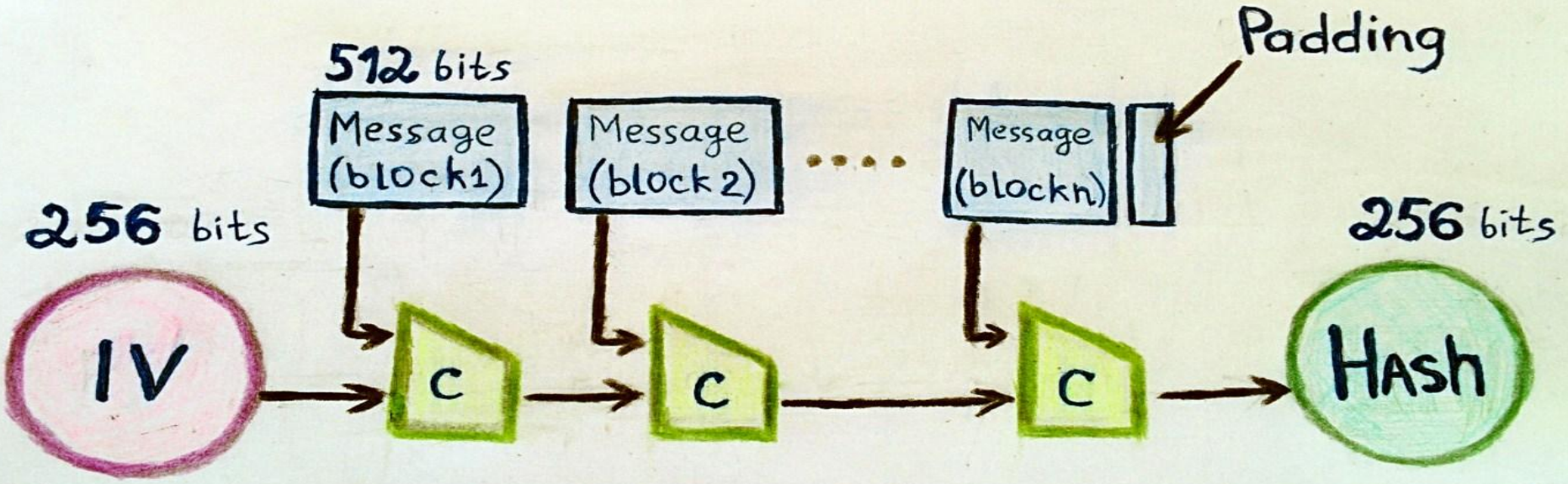
Automatic Repeat ReQuest

- Basic mechanism for fault-tolerance & redundancy
- Echoes back each message received
- Can be used to move two parties in lock-step or parallel fashion


Receiver(s) start up first, then the sender, each holding resources
File content is received (ARQ) and written (both) in order
Indicates success or error, releases resources and exits
Startup
Send/Recv Metadata
Send/Recv File Content
Integrity Check
Clean Exit
Metadata is guaranteed to be sent before file content
SHA-2 256-bit hashing algorithm verifies data after transmission


Input
Digest
Fox
cryptographic hash function
DFCD 3454 BBEA 788A 751A 696C 24D9 7009 CA99 2D17
The red fox jumps over the blue dog
cryptographic hash function
0086 46BB FB7D CBE2 823C ACC7 6CD1 90B1 EE6E 3ABC
The red fox jumps ouer the blue dog
cryptographic hash function
8FD8 7558 7851 4F32 D1C6 76B1 79A9 0DA4 AEFE 4819
The red fox jumps oevr the blue dog
cryptographic hash function
FCD3 7FDB 5AF2 C6FF 915F D401 C0A9 7D9A 46AF FB45
The red fox jumps oer the blue dog
cryptographic hash function
8ACA D682 D588 4C75 4BF4 1799 7D88 BCF8 92B9 6A6C

SHA-256


512 bits
Message (block1)
Message (block 2)
Message (blockn)
Padding
256 bits
IV
C
C
C
256 bits
Hash

Unicast vs. Multicast

- Simple app. and UDP `SendFile` app. both designed with multicast support
- Feedback handling however is indeterminate for “one-to-many” file transfers
- Sender is not aware of number of receivers

Documentation

General Overview of Implementations

- File structure
- Brief descriptions of functions
- Packet & feedback types
- Fragmentation & reassembly
- Timers & delays
- Data integrity
- Transmission procedure
- Limitations addressed
- Issues

UDP SendFile App Documentation

GENERAL INFORMATION

1. Implementations

There are two separate implementations: A "Fail-Fast" version under UDPSendFileApp and a lightweight "ARQ" version under UDPSendFileApp/ARQ_version. Both versions make use of timers and feedback but the method and nature of use is significantly different.

In the Fail-Fast version, the sender is meant to make minimal use of feedback within a protocol-defined time limit and otherwise prefer to terminate. Neither sender nor receiver uses any feedback for file content transmission and its use is reserved for metadata transmission and a final file content integrity check.

In the lightweight ARQ version, both the sender and receivers communicate with each other through the feedback mechanism for metadata, file content transmission and the final content integrity check. A positive acknowledgment feedback is expected by the sender and sent by receivers for each segment of data transmitted and received in order, whereas a negative acknowledgment from receiver to sender or simply the lack of positive feedback induces repeated transmission of the same segment.

Note: All of the following apply to both implementations unless otherwise noted.

2. File structure

- **mcastsendfile.py**: This is the file run by the "sender" host.
- **mcastrecvfile.py**: This is the file run by "receiver" hosts(s).
- **udpsendfile_common.py**: This file includes constants and functions used by both of the above files.

In any given run, there is only one sender host but there could be more than one receiver host. The application has been designed to run on Python 3.7.x based on what is available on the VMs. The sender and receiver(s) could be on the same machine or different ones. Testing has been performed in both scenarios.

3. Brief descriptions of major functions by file

(Descriptions are based on the source code comments)

Testing Setup

Automated:

- UNIX shell (Bash) scripts that run receiver then sender for <N> number of times
- Makes sure rogue processes are killed so that port access errors are minimized
- Indicates number of failures out of total runs & elapsed time (for all runs of new UDP SendFile App)

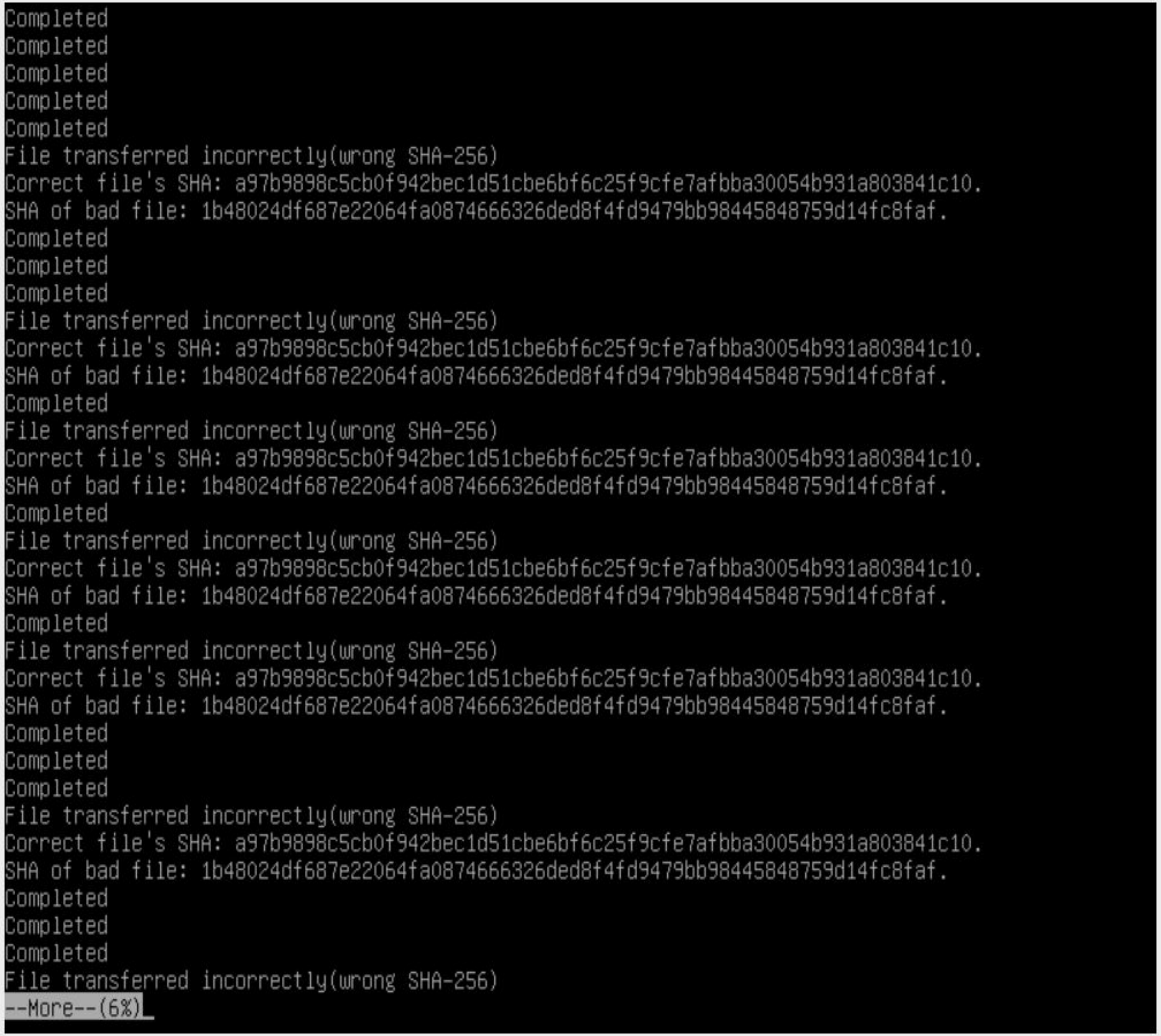
Manual: Hands-on and using ScaPy3

```
        let num_of_failures="$num_of_failures"+1
    fi

    # Extract current process to be able to continue to the next iteration:
    # Miriam Briskman, 10/23
    prev_python_prog_id=$(ps -af | grep "[m]cast" | awk '{print $2}')

    # Repeat the termination of a prev running python program:
    # Miriam Briskman, 10/23
    if [ ! -z "$prev_python_prog_id" ]
    then
        kill -9 $prev_python_prog_id
    fi
done

echo "Rate of failure: $num_of_failures / $total_num_of_runs. Please view \"$OUT_FILE\" for both out
put and errors."

# EOF.
"testing_script_sample_old_app.sh" 76L, 3147C written
brooklyn@brooklyn:~/proj1sendfile-bkr/Experiments$ ./testing_script_sample_old_app.sh 10 lorem_ipsum
_1000_bytes.txt
./testing_script_sample_old_app.sh: line 37: 15673 Killed                  python ../SimpleMCastSend
File/mcastrecvfile_miriam.py $ip_address 224.1.1.5 50000 >> $OUT_FILE 2>&1
./testing_script_sample_old_app.sh: line 37: 15715 Killed                  python ../SimpleMCastSend
File/mcastrecvfile_miriam.py $ip_address 224.1.1.5 50000 >> $OUT_FILE 2>&1
./testing_script_sample_old_app.sh: line 37: 15729 Killed                  python ../SimpleMCastSend
File/mcastrecvfile_miriam.py $ip_address 224.1.1.5 50000 >> $OUT_FILE 2>&1
./testing_script_sample_old_app.sh: line 37: 15757 Killed                  python ../SimpleMCastSend
File/mcastrecvfile_miriam.py $ip_address 224.1.1.5 50000 >> $OUT_FILE 2>&1
./testing_script_sample_old_app.sh: line 37: 15771 Killed                  python ../SimpleMCastSend
File/mcastrecvfile_miriam.py $ip_address 224.1.1.5 50000 >> $OUT_FILE 2>&1
./testing_script_sample_old_app.sh: line 37: 15785 Killed                  python ../SimpleMCastSend
File/mcastrecvfile_miriam.py $ip_address 224.1.1.5 50000 >> $OUT_FILE 2>&1
Rate of failure: 0 / 10. Please view "./testing_output_old_app.txt" for both output and errors.
brooklyn@brooklyn:~/proj1sendfile-bkr/Experiments$
```

Testing of the **old** application

File size:
1,000 bytes

Rate of failure:
0 out of 10



```
Completed
Completed
Completed
Completed
Completed
File transferred incorrectly(wrong SHA-256)
Correct file's SHA: a97b9898c5cb0f942bec1d51cbe6bf6c25f9cfe7afbba30054b931a803841c10.
SHA of bad file: 1b48024df687e22064fa0874666326ded8f4fd9479bb98445848759d14fc8faf.
Completed
Completed
Completed
File transferred incorrectly(wrong SHA-256)
Correct file's SHA: a97b9898c5cb0f942bec1d51cbe6bf6c25f9cfe7afbba30054b931a803841c10.
SHA of bad file: 1b48024df687e22064fa0874666326ded8f4fd9479bb98445848759d14fc8faf.
Completed
File transferred incorrectly(wrong SHA-256)
Correct file's SHA: a97b9898c5cb0f942bec1d51cbe6bf6c25f9cfe7afbba30054b931a803841c10.
SHA of bad file: 1b48024df687e22064fa0874666326ded8f4fd9479bb98445848759d14fc8faf.
Completed
File transferred incorrectly(wrong SHA-256)
Correct file's SHA: a97b9898c5cb0f942bec1d51cbe6bf6c25f9cfe7afbba30054b931a803841c10.
SHA of bad file: 1b48024df687e22064fa0874666326ded8f4fd9479bb98445848759d14fc8faf.
Completed
File transferred incorrectly(wrong SHA-256)
Correct file's SHA: a97b9898c5cb0f942bec1d51cbe6bf6c25f9cfe7afbba30054b931a803841c10.
SHA of bad file: 1b48024df687e22064fa0874666326ded8f4fd9479bb98445848759d14fc8faf.
Completed
Completed
Completed
File transferred incorrectly(wrong SHA-256)
Correct file's SHA: a97b9898c5cb0f942bec1d51cbe6bf6c25f9cfe7afbba30054b931a803841c10.
SHA of bad file: 1b48024df687e22064fa0874666326ded8f4fd9479bb98445848759d14fc8faf.
Completed
Completed
Completed
File transferred incorrectly(wrong SHA-256)
--More--(6%)
```

Testing of the **old** application

File size:
10,000 bytes

Rate of failure:
98 out of 100

Note: the differences in checksum values (added in testing script) of the sent and received files indicates **data corruption**

```
Traceback (most recent call last):
  File "../SimpleMCastSendFile/mcastsendfile_miriam.py", line 77, in <module>
    main(sys.argv)
  File "../SimpleMCastSendFile/mcastsendfile_miriam.py", line 73, in main
    mc_send_file(hostipaddr, mcgrpipaddr, mcport, filename)
  File "../SimpleMCastSendFile/mcastsendfile_miriam.py", line 31, in mc_send_file
    sender.sendto(contents, endpoint) # Already in a form of a byte string
OSError: [Errno 90] Message too long
Traceback (most recent call last):
  File "../SimpleMCastSendFile/mcastsendfile_miriam.py", line 77, in <module>
    main(sys.argv)
  File "../SimpleMCastSendFile/mcastsendfile_miriam.py", line 73, in main
    mc_send_file(hostipaddr, mcgrpipaddr, mcport, filename)
  File "../SimpleMCastSendFile/mcastsendfile_miriam.py", line 31, in mc_send_file
    sender.sendto(contents, endpoint) # Already in a form of a byte string
OSError: [Errno 90] Message too long
Traceback (most recent call last):
  File "../SimpleMCastSendFile/mcastsendfile_miriam.py", line 77, in <module>
    main(sys.argv)
  File "../SimpleMCastSendFile/mcastsendfile_miriam.py", line 73, in main
    mc_send_file(hostipaddr, mcgrpipaddr, mcport, filename)
  File "../SimpleMCastSendFile/mcastsendfile_miriam.py", line 31, in mc_send_file
    sender.sendto(contents, endpoint) # Already in a form of a byte string
OSError: [Errno 90] Message too long
Traceback (most recent call last):
  File "../SimpleMCastSendFile/mcastsendfile_miriam.py", line 77, in <module>
    main(sys.argv)
  File "../SimpleMCastSendFile/mcastsendfile_miriam.py", line 73, in main
    mc_send_file(hostipaddr, mcgrpipaddr, mcport, filename)
  File "../SimpleMCastSendFile/mcastsendfile_miriam.py", line 31, in mc_send_file
    sender.sendto(contents, endpoint) # Already in a form of a byte string
OSError: [Errno 90] Message too long
Traceback (most recent call last):
  File "../SimpleMCastSendFile/mcastsendfile_miriam.py", line 77, in <module>
    main(sys.argv)
  File "../SimpleMCastSendFile/mcastsendfile_miriam.py", line 73, in main
--More--(4%)
```

Testing of the **old** application

File size:
100,000 bytes

Rate of failure:
100 out of 100

Note: the program itself fails for files of this size

```
brooklyn@brooklyn: ~/proj1sendfile-bkr/Experiments
remote: Compressing objects: 100% (2/2), done.
remote: Total 5 (delta 3), reused 5 (delta 3), pack-reused 0
Unpacking objects: 100% (5/5), done.
From https://github.com/cunychenhclass/proj1sendfile-bkr
   8a05ba0..181e88b  master     -> origin/master
Updating 8a05ba0..181e88b
Fast-forward
 Experiments/files_of_sender/lorem_ipsum_10000_bytes.txt | 34 ++++++++++++++++++++++++++++++++++--
 Experiments/lorem_ipsum_10000_bytes.txt                 | 34 ++++++++++++++++++++++++++++++++++--
 2 files changed, 64 insertions(+), 4 deletions(-)
brooklyn@brooklyn:~/proj1sendfile-bkr$ ls
Experiments  Readme.md  Report  SimpleMCastSendFile  UDPSendFileApp  UDPSendFileProto
brooklyn@brooklyn:~/proj1sendfile-bkr$ cd Experiments/
brooklyn@brooklyn:~/proj1sendfile-bkr/Experiments$ ls
 files_of_sender     lorem_ipsum_100000_bytes.txt   lorem_ipsum_100_bytes.txt    testing_script_ARQ.sh
'HelloWorld!'$'\n'   lorem_ipsum_10000_bytes.txt    testing_output_new_app.txt   testing_script_sample_new_app.sh
 helloWorld.txt      lorem_ipsum_1000_bytes.txt     testing_output_old_app.txt   testing_script_sample_old_app.sh
brooklyn@brooklyn:~/proj1sendfile-bkr/Experiments$ ./testing_script_sample_new_app.sh 100 lorem_ipsum_10000_bytes.txt
Rate of failure: 0 / 100. Please view "./testing_output_new_app.txt" for both output and errors.
Elapsed time: 0 minutes, 13 seconds
brooklyn@brooklyn:~/proj1sendfile-bkr/Experiments$ ./testing_script_sample_new_app.sh 1000 lorem_ipsum_10000_bytes.txt
Rate of failure: 1 / 1000. Please view "./testing_output_new_app.txt" for both output and errors.
Elapsed time: 2 minutes, 15 seconds
brooklyn@brooklyn:~/proj1sendfile-bkr/Experiments$ more testing_output_new_app.txt
Completed: received lorem_ipsum_10000_bytes.txt

Filename received: lorem_ipsum_10000_bytes.txt
Data size received: 10033
Checksum received: 1b48024df687e22064fa0874666326ded8f4fd9479bb98445848759d14fc8faf
Received file packet number: 1
Received file packet number: 2
Received file packet number: 3
Received file packet number: 4
```

Testing of the **new** application

(**Fail-Fast** version)

File size:
10,000 bytes

Rate of failure:
1 / 1000?

```
brooklyn@brooklyn: ~/proj1sendfile-bkr/Experiments
...skipping
Checksum feedback: A
Final feedback: D
Timed out, aborting...
Sending file: lorem_ipsum_10000_bytes.txt

Filename sent: b'1lorem_ipsum_10000_bytes.txt'
Aborting due to sender request...
Completed: received lorem_ipsum_10000_bytes.txt

Filename received: lorem_ipsum_10000_bytes.txt
Data size received: 10033
Checksum received: 1b48024df687e22064fa0874666326ded8f4fd9479bb98445848759d14fc8faf
Received file packet number: 1
Received file packet number: 2
Received file packet number: 3
Received file packet number: 4
Received file packet number: 5
Received file packet number: 6
Received file packet number: 7
Received file packet number: 8
Received file packet number: 9
Received file packet number: 10
Checksum of local data: 1b48024df687e22064fa0874666326ded8f4fd9479bb98445848759d14fc8faf

Completed: sent lorem_ipsum_10000_bytes.txt
Sending file: lorem_ipsum_10000_bytes.txt

Filename sent: b'1lorem_ipsum_10000_bytes.txt'
Filename feedback: A
Data size sent: b'210033'
Data size feedback: A
brooklyn@brooklyn:~/proj1sendfile-bkr/Experiments$
```

Time out error due to slower startup of the receiver relative to the sender

Fail-fast version did exactly **what it was designed to** when sender did not receive feedback from receiver within time limit

```
if [ ! -z "$prev_python_prog_id" ]
then
    kill -9 $prev_python_prog_id
fi

SECONDS=0
for i in $(seq 1 "$total_num_of_runs")
do
    python ../UDPSendFileApp/mcastrecvfile.py -i $ip_address 224.1.1.5 50000 50001 >> $OUT_FILE 2>&1 &
    sleep 0.02
    python ../UDPSendFileApp/mcastsendfile.py $ip_address 224.1.1.5 50000 50001 "$filename" >> $OUT_FILE 2>&1

    if [ $? -ne 0 ]; then
        let num_of_failures="$num_of_failures"+1
    fi

    # Extract current process to be able to continue to the next iteration:
    # Miriam Briskman, 10/23
    prev_python_prog_id=$(ps -af | grep "[m]cast" | awk '{print $2}')

    # Repeat the termination of a prev running python program:
    # Miriam Briskman, 10/23
    if [ ! -z "$prev_python_prog_id" ]
    then
        kill -9 $prev_python_prog_id
    fi
done
elapsed=$SECONDS

echo "Rate of failure: $num_of_failures / $total_num_of_runs. Please view \"$OUT_FILE\" for both output and errors."
echo "Elapsed time: $(($elapsed / 60)) minutes, $(($elapsed % 60)) seconds"

# EOF.
~
~
~
testing_script_sample_new_app.sh                                    64,1          Bot
search hit BOTTOM, continuing at TOP
```

Correct delay between starting receiver and sender is an estimate

Our tests indicate 20 ms or higher is generally enough but exceptions remain

Of course, this is **not a concern for manual testing or use**

```
brooklyn@brooklyn: ~/proj1sendfile-bkr/Experiments
Username for 'https://github.com': olatif@u.rochester.edu
Password for 'https://olatif@u.rochester.edu@github.com':
Enumerating objects: 11, done.
Counting objects: 100% (11/11), done.
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 555 bytes | 555.00 KiB/s, done.
Total 6 (delta 5), reused 0 (delta 0)
remote: Resolving deltas: 100% (5/5), completed with 5 local objects.
To https://github.com/cunychenhclass/proj1sendfile-bkr
   d8cf8e1..8a05ba0  master -> master
brooklyn@brooklyn:~/proj1sendfile-bkr$ ls
Experiments  Readme.md  Report  SimpleMCastSendFile  UDPSendFileApp  UDPSendFileProto
brooklyn@brooklyn:~/proj1sendfile-bkr$ cd Experiments/
brooklyn@brooklyn:~/proj1sendfile-bkr/Experiments$ ls
 files_of_sender     lorem_ipsum_100000_bytes.txt   lorem_ipsum_100_bytes.txt    testing_script_ARQ.sh
'HelloWorld!'$'\n'   lorem_ipsum_10000_bytes.txt    testing_output_new_app.txt   testing_script_sample_new_app.sh
 helloWorld.txt      lorem_ipsum_1000_bytes.txt     testing_output_old_app.txt   testing_script_sample_old_app.sh
brooklyn@brooklyn:~/proj1sendfile-bkr/Experiments$ ./testing_script_sample_new_app.sh 1000 lorem_ipsum_1000_bytes.txt
Rate of failure: 0 / 1000. Please view "./testing_output_new_app.txt" for both output and errors.
Elapsed time: 1 minutes, 24 seconds
brooklyn@brooklyn:~/proj1sendfile-bkr/Experiments$ ./testing_script_sample_new_app.sh 10 lorem_ipsum_1000_bytes.txt
Rate of failure: 0 / 10. Please view "./testing_output_new_app.txt" for both output and errors.
Elapsed time: 0 minutes, 1 seconds
brooklyn@brooklyn:~/proj1sendfile-bkr/Experiments$ ./testing_script_sample_new_app.sh 10 lorem_ipsum_100_bytes.txt
Rate of failure: 0 / 10. Please view "./testing_output_new_app.txt" for both output and errors.
Elapsed time: 0 minutes, 1 seconds
brooklyn@brooklyn:~/proj1sendfile-bkr/Experiments$ ./testing_script_sample_new_app.sh 10 lorem_ipsum_100000_bytes.txt
Rate of failure: 0 / 10. Please view "./testing_output_new_app.txt" for both output and errors.
Elapsed time: 0 minutes, 6 seconds
brooklyn@brooklyn:~/proj1sendfile-bkr/Experiments$ ./testing_script_sample_new_app.sh 100 lorem_ipsum_100000_bytes.txt
Rate of failure: 0 / 100. Please view "./testing_output_new_app.txt" for both output and errors.
Elapsed time: 1 minutes, 0 seconds
brooklyn@brooklyn:~/proj1sendfile-bkr/Experiments$
```

Testing of the **new** application

(**Fail-Fast** version)

File sizes:
100, 1K, 10K, 100K bytes

Rate of failure:
0 out of 10, 100, 1000

brooklyn@brooklyn: ~/proj1sendfile-bkr/Experiments

```
Filename sent: b'1lorem_ipsum_10000_bytes.txt'
Aborting due to sender request...
Completed: received lorem_ipsum_10000_bytes.txt

Filename received: lorem_ipsum_10000_bytes.txt
Data size received: 10033
Checksum received: 1b48024df687e22064fa0874666326ded8f4fd9479bb98445848759d14fc8faf
Received file packet number: 1
Received file packet number: 2
Received file packet number: 3
Received file packet number: 4
Received file packet number: 5
Received file packet number: 6
Received file packet number: 7
Received file packet number: 8
Received file packet number: 9
Received file packet number: 10
Checksum of local data: 1b48024df687e22064fa0874666326ded8f4fd9479bb98445848759d14fc8faf

Completed: sent lorem_ipsum_10000_bytes.txt
Sending file: lorem_ipsum_10000_bytes.txt

Filename sent: b'1lorem_ipsum_10000_bytes.txt'
Filename feedback: A
Data size sent: b'210033'
Data size feedback: A
brooklyn@brooklyn:~/proj1sendfile-bkr/Experiments$ ./testing_script_ARQ.sh 100 lorem_ipsum_10000_bytes.txt
Rate of failure: 0 / 100. Please view "./testing_output_new_app.txt" for both output and errors.
Elapsed time: 0 minutes, 13 seconds
brooklyn@brooklyn:~/proj1sendfile-bkr/Experiments$ ./testing_script_ARQ.sh 1000 lorem_ipsum_10000_bytes.txt
Rate of failure: 0 / 1000. Please view "./testing_output_new_app.txt" for both output and errors.
Elapsed time: 2 minutes, 10 seconds
brooklyn@brooklyn:~/proj1sendfile-bkr/Experiments$
```

Testing of the **new** application

(**ARQ** version)

File size:
10,000 bytes

Rate of failure:
0 / 1000

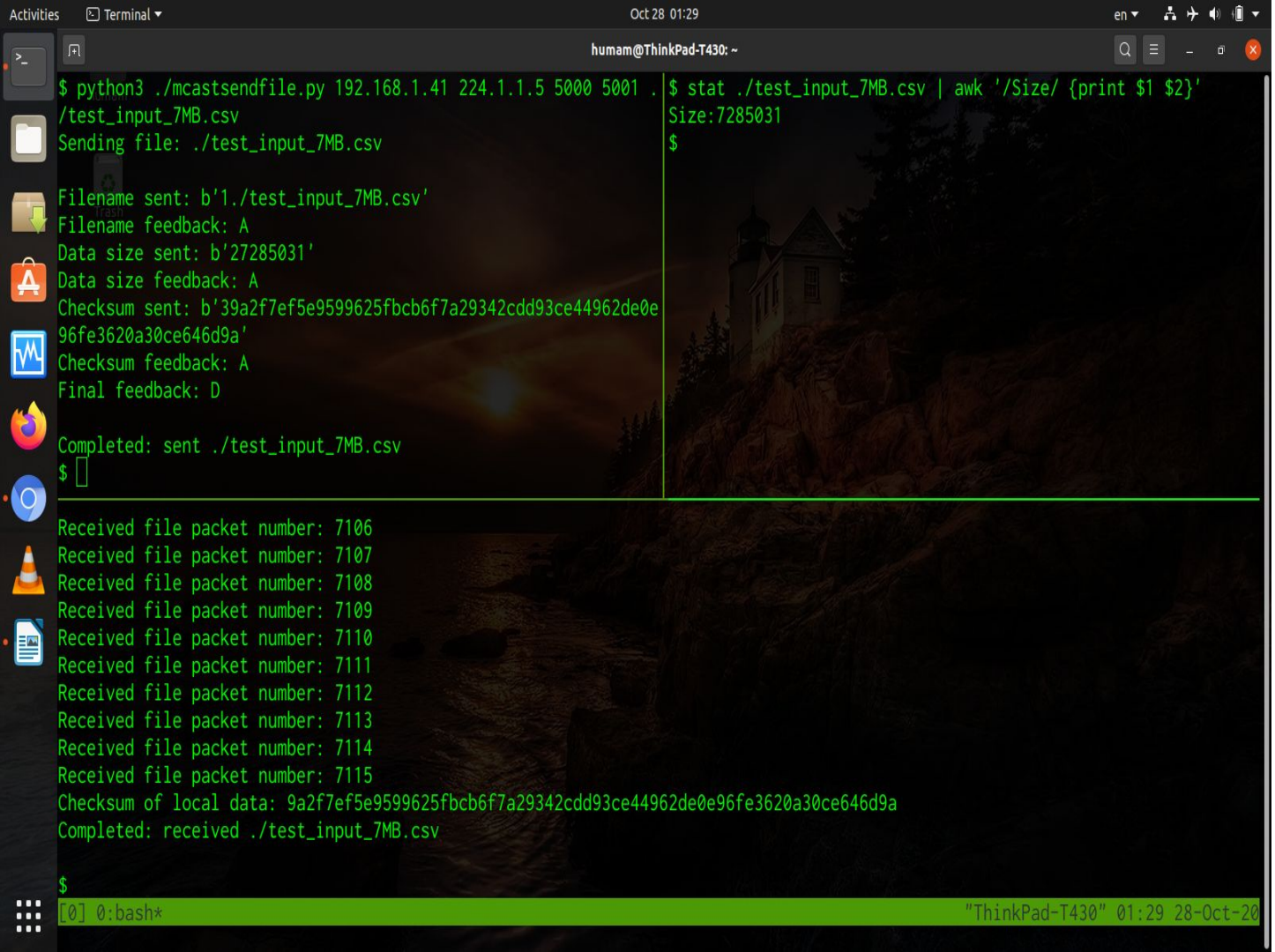
```
brooklyn@brooklyn: ~/proj1sendfile-bkr/Experiments
 helloWorld.txt      lorem_ipsum_1000_bytes.txt     testing_output_old_app.txt   testing_script_sample_old_app.sh
brooklyn@brooklyn:~/proj1sendfile-bkr/Experiments$ ./testing_script_sample_new_app.sh 1000 lorem_ipsum_1000_bytes.txt
Rate of failure: 0 / 1000. Please view "./testing_output_new_app.txt" for both output and errors.
Elapsed time: 1 minutes, 24 seconds
brooklyn@brooklyn:~/proj1sendfile-bkr/Experiments$ ./testing_script_sample_new_app.sh 10 lorem_ipsum_1000_bytes.txt
Rate of failure: 0 / 10. Please view "./testing_output_new_app.txt" for both output and errors.
Elapsed time: 0 minutes, 1 seconds
brooklyn@brooklyn:~/proj1sendfile-bkr/Experiments$ ./testing_script_sample_new_app.sh 10 lorem_ipsum_100_bytes.txt
Rate of failure: 0 / 10. Please view "./testing_output_new_app.txt" for both output and errors.
Elapsed time: 0 minutes, 1 seconds
brooklyn@brooklyn:~/proj1sendfile-bkr/Experiments$ ./testing_script_sample_new_app.sh 10 lorem_ipsum_100000_bytes.txt
Rate of failure: 0 / 10. Please view "./testing_output_new_app.txt" for both output and errors.
Elapsed time: 0 minutes, 6 seconds
brooklyn@brooklyn:~/proj1sendfile-bkr/Experiments$ ./testing_script_sample_new_app.sh 100 lorem_ipsum_100000_bytes.txt
Rate of failure: 0 / 100. Please view "./testing_output_new_app.txt" for both output and errors.
Elapsed time: 1 minutes, 0 seconds
brooklyn@brooklyn:~/proj1sendfile-bkr/Experiments$ ls
 files_of_sender     lorem_ipsum_100000_bytes.txt   lorem_ipsum_100_bytes.txt    testing_script_ARQ.sh
'HelloWorld!'$'\n'   lorem_ipsum_10000_bytes.txt    testing_output_new_app.txt   testing_script_sample_new_app.sh
 helloWorld.txt      lorem_ipsum_1000_bytes.txt     testing_output_old_app.txt   testing_script_sample_old_app.sh
brooklyn@brooklyn:~/proj1sendfile-bkr/Experiments$ ./testing_script_ARQ.sh 100 lorem_ipsum_1000_bytes.txt
Rate of failure: 0 / 100. Please view "./testing_output_new_app.txt" for both output and errors.
Elapsed time: 0 minutes, 13 seconds
brooklyn@brooklyn:~/proj1sendfile-bkr/Experiments$ ./testing_script_ARQ.sh 1000 lorem_ipsum_1000_bytes.txt
Rate of failure: 0 / 1000. Please view "./testing_output_new_app.txt" for both output and errors.
Elapsed time: 1 minutes, 21 seconds
brooklyn@brooklyn:~/proj1sendfile-bkr/Experiments$ ./testing_script_ARQ.sh 10 lorem_ipsum_100000_bytes.txt
Rate of failure: 0 / 10. Please view "./testing_output_new_app.txt" for both output and errors.
Elapsed time: 0 minutes, 6 seconds
brooklyn@brooklyn:~/proj1sendfile-bkr/Experiments$ ./testing_script_ARQ.sh 100 lorem_ipsum_100000_bytes.txt
Rate of failure: 0 / 100. Please view "./testing_output_new_app.txt" for both output and errors.
Elapsed time: 1 minutes, 1 seconds
brooklyn@brooklyn:~/proj1sendfile-bkr/Experiments$
```

Testing of the **new** application

(**ARQ** version)

File sizes:
100, 1K, 10K, 100K bytes

Rate of failure:
0 out of 10, 100, 1000

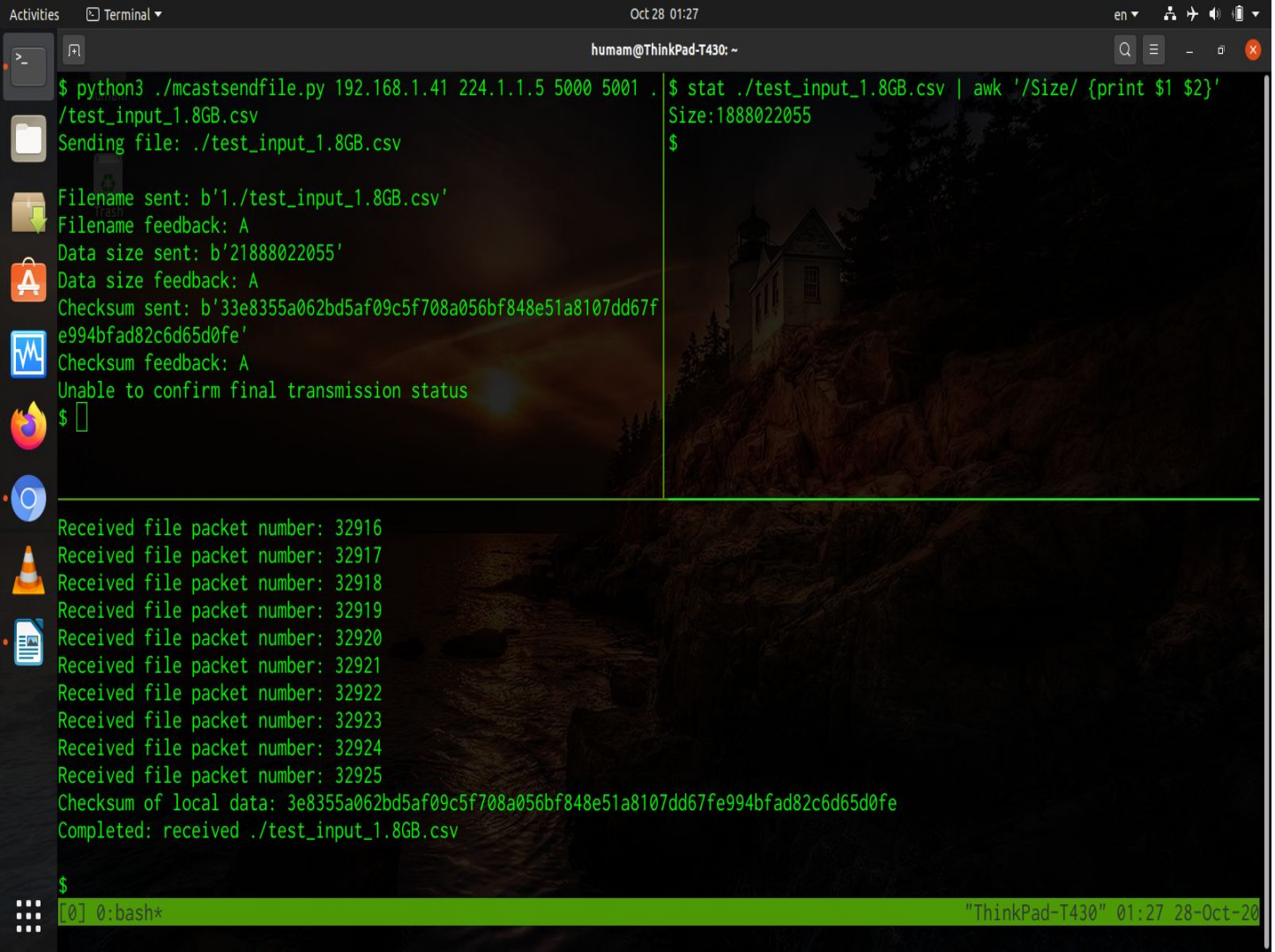


Testing of the **new** application

(Manual, **Fail-Fast** version, Buffered file writing)

File size:
7 MB

Result:
Success



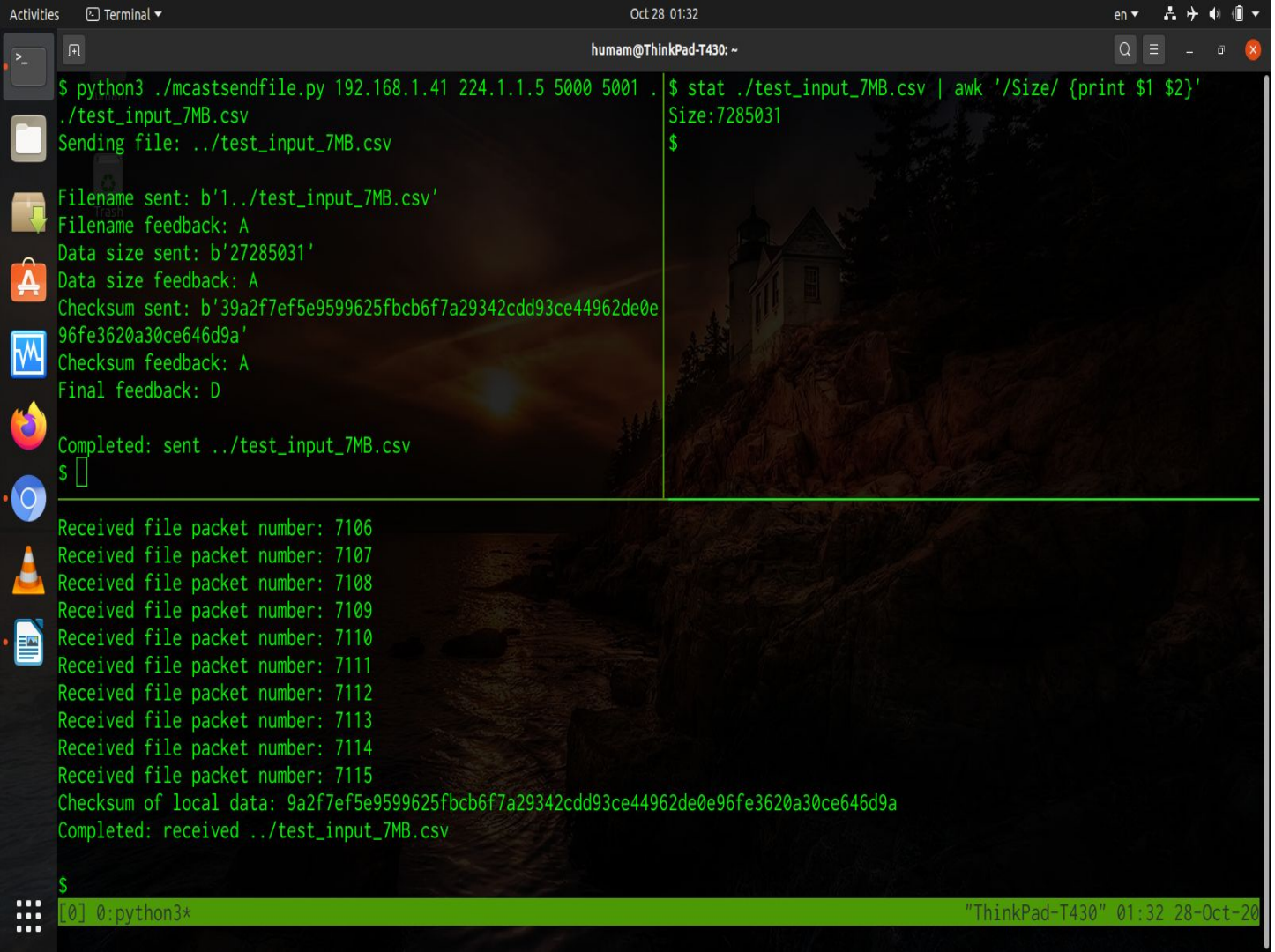
Testing of the **new** application

(Manual, **Fail-Fast** version, Buffered file writing)

File size:
1.8 GB

Result:
Success

Time saving cheat:
56kB block size

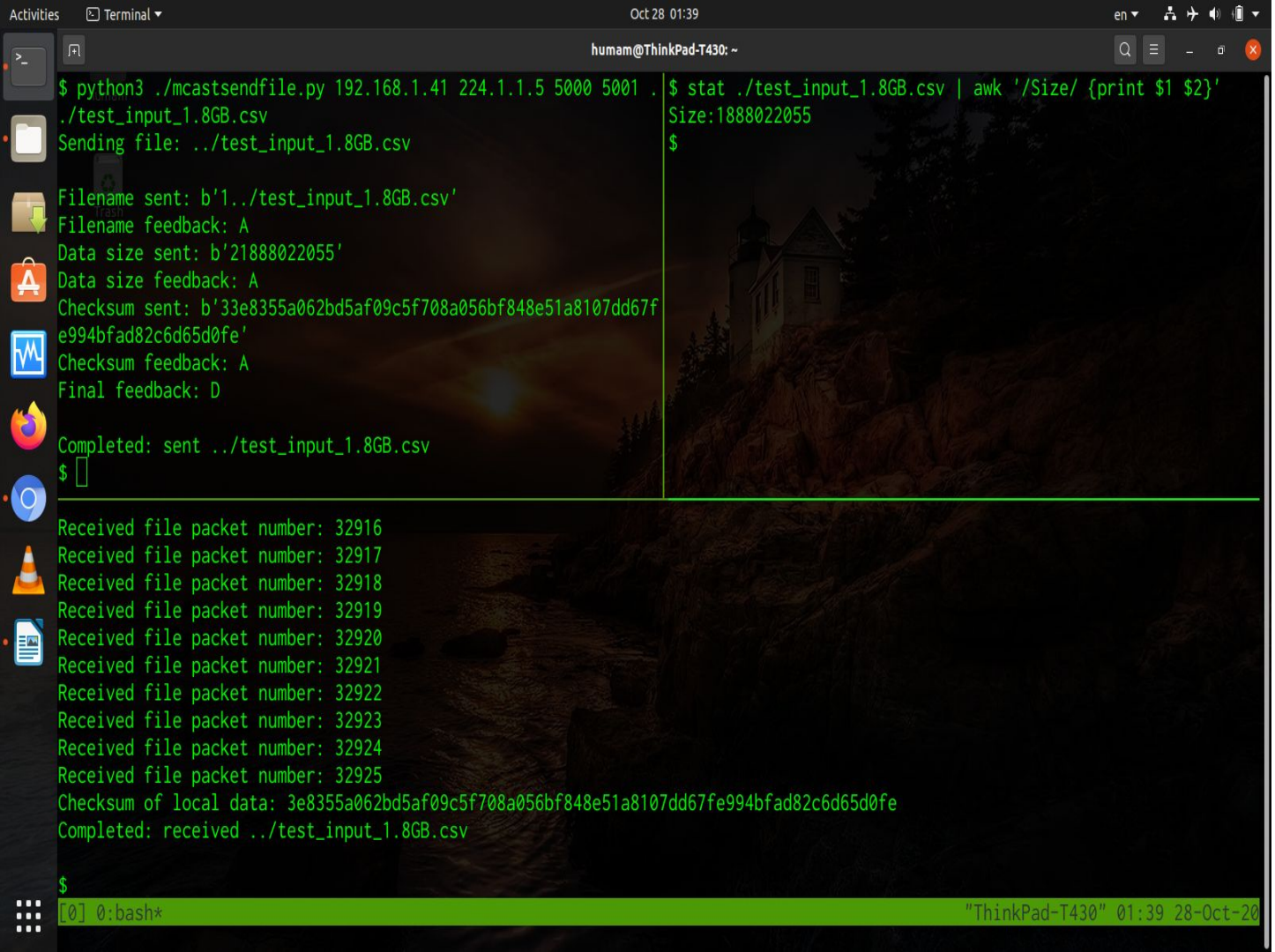


Testing of the **new** application

(Manual, **ARQ** version, Incremental file writing)

File size:
7 MB

Result:
Success



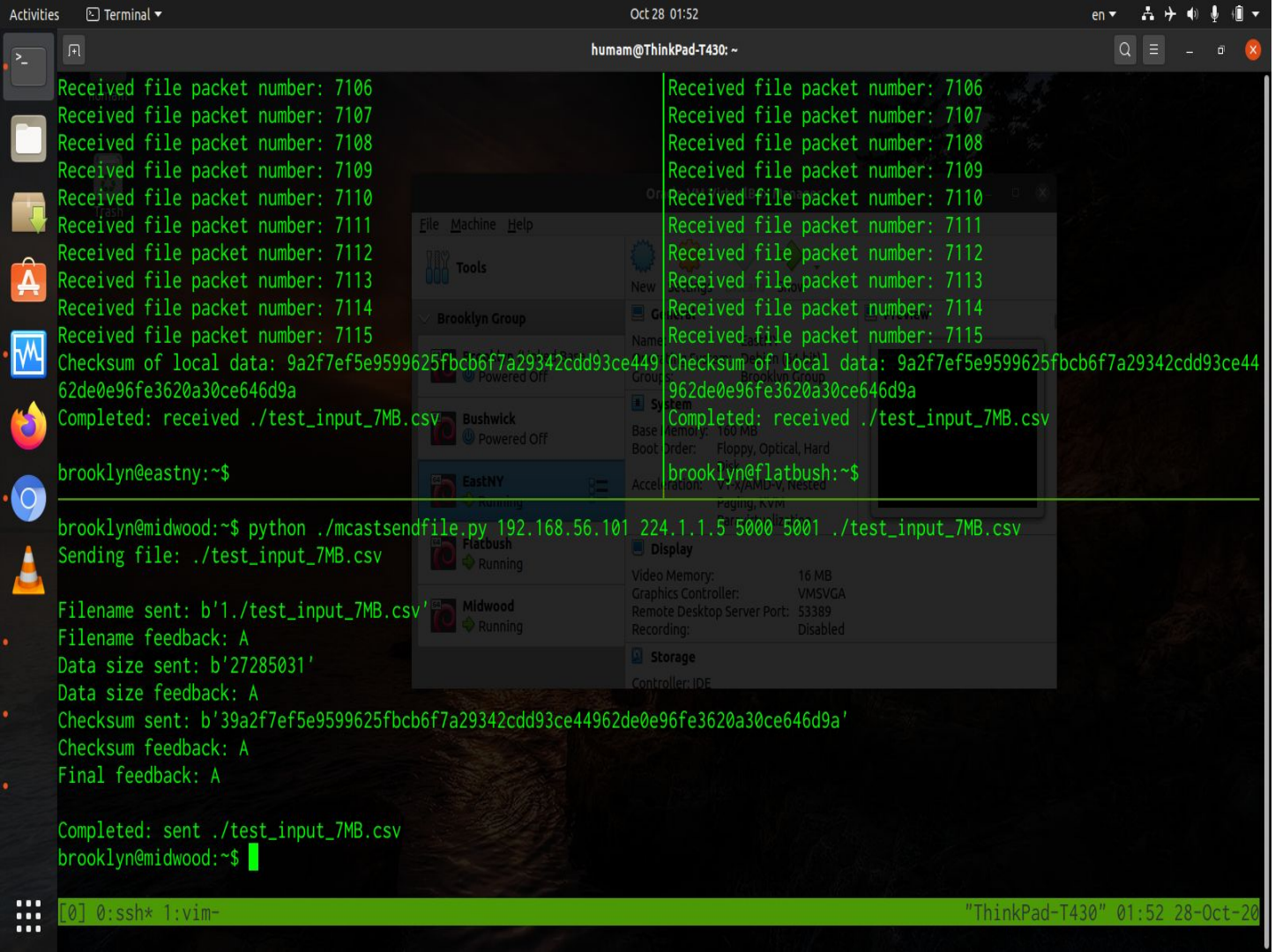
Testing of the **new** application

(Manual, **ARQ** version,
Incremental file writing)

File size:
1.8 GB

Result:
Success

Time saving cheat:
56kB block size



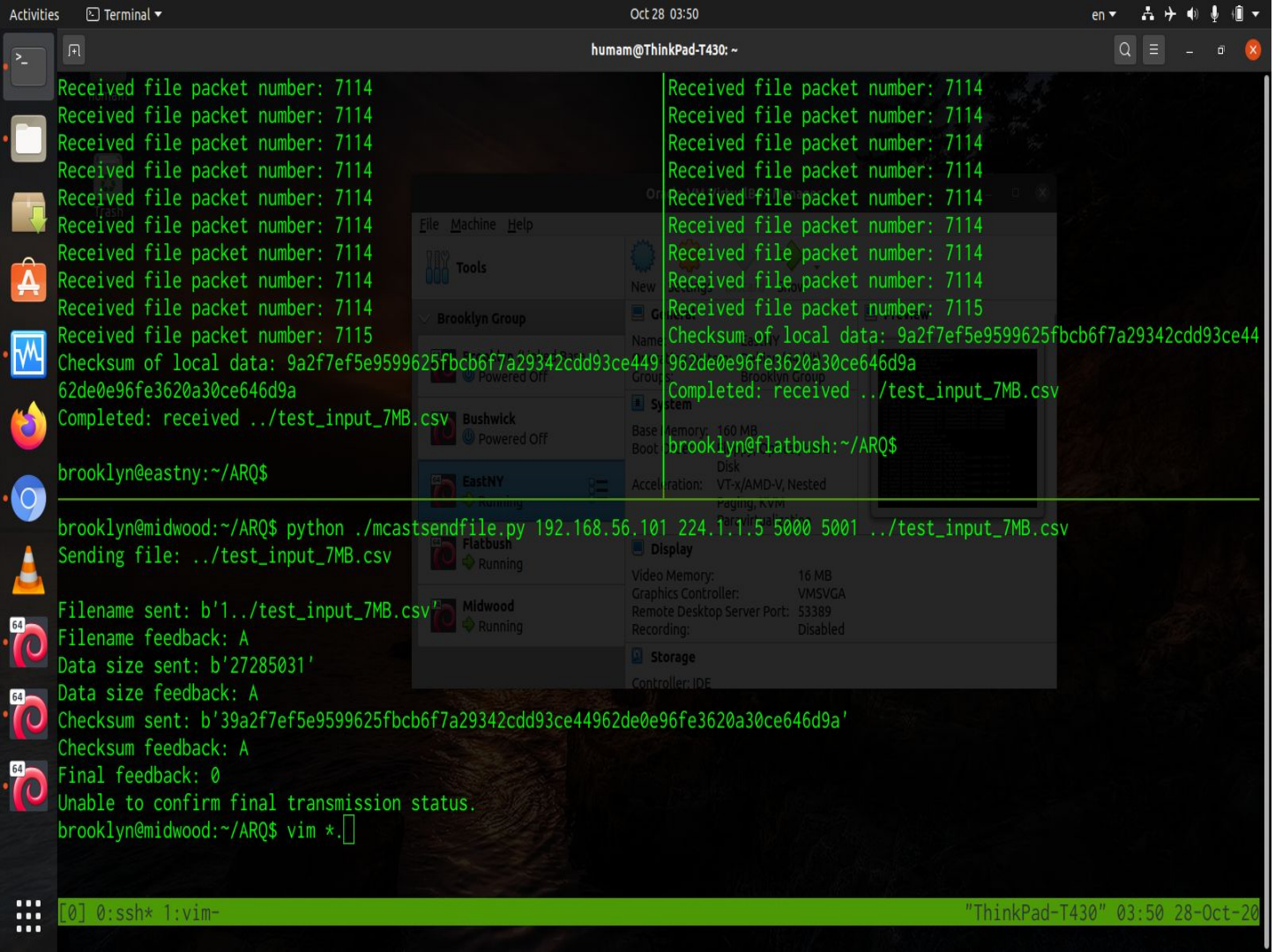
Testing of the **new** application

(Manual, **Fail-Fast** version, Multiple receivers)

File size:
7 MB

Result:
Success

Feedback handling is indeterminate



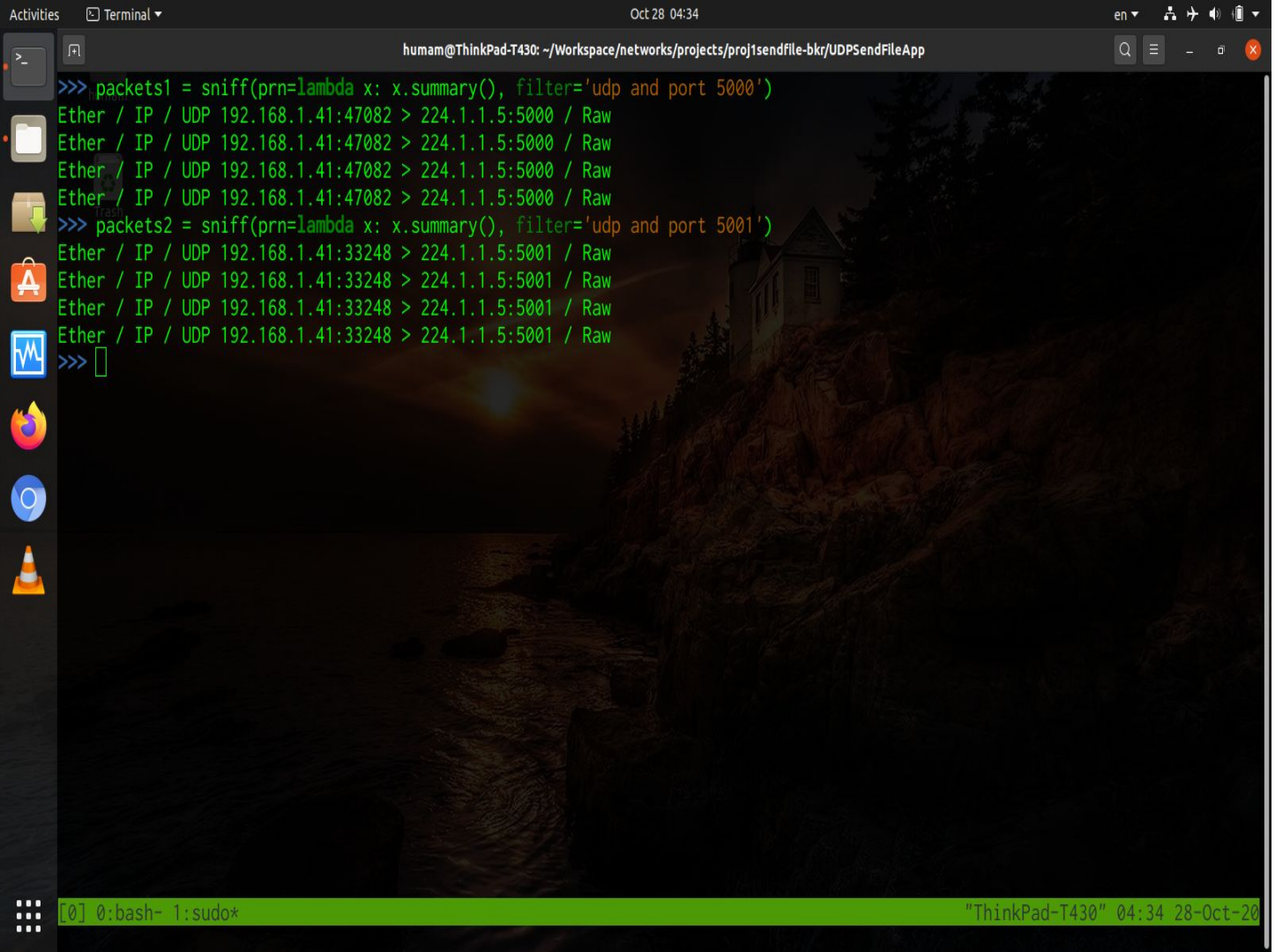
Testing of the **new** application

(Manual, **ARQ** version, Multiple receivers)

File size:
7 MB

Result:
Success

Feedback handling is indeterminate

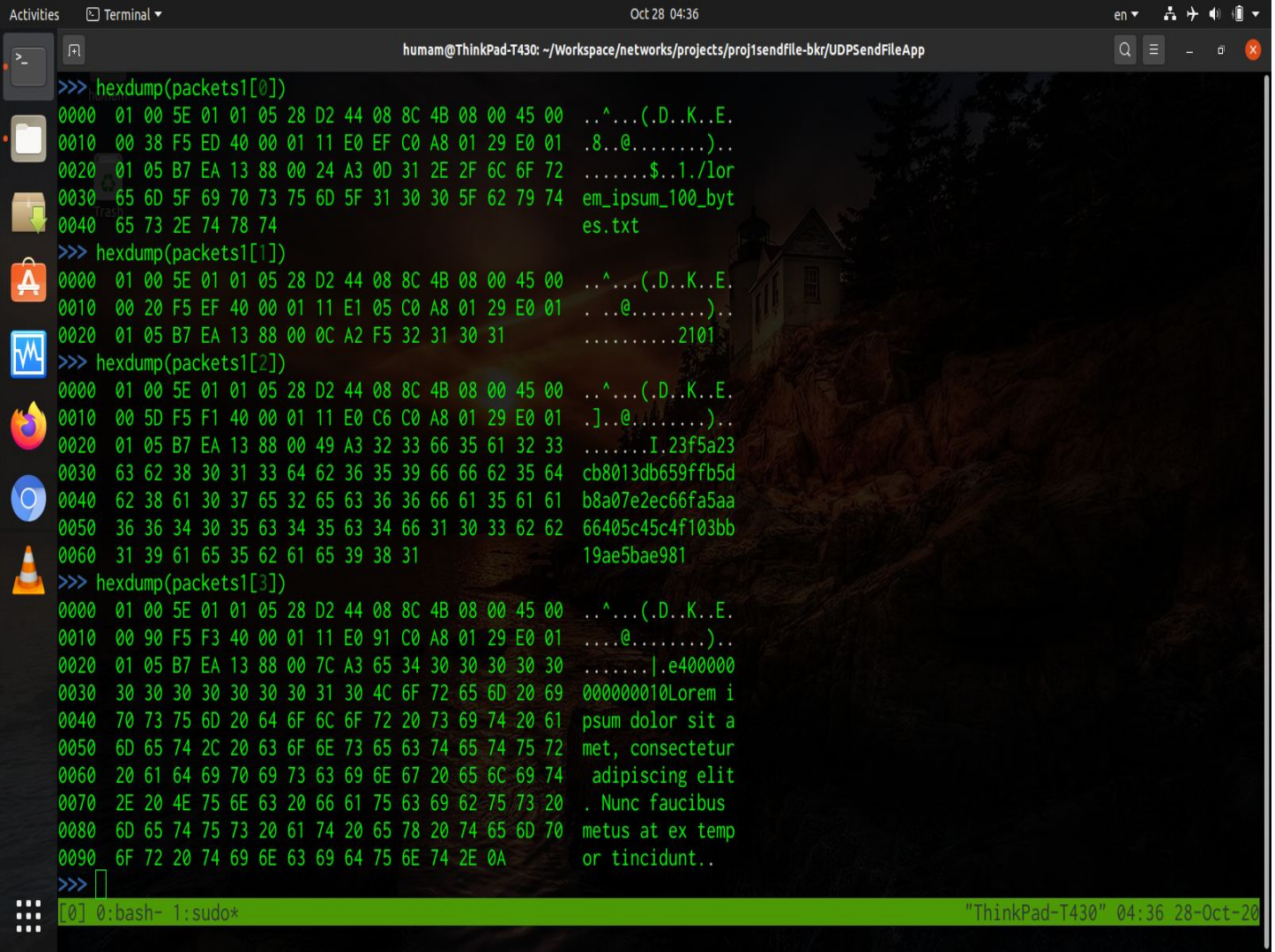


Testing of the **new** application

(manual, **Fail-Fast** version)

File size:
100 bytes

Result:
ScaPy detects **4 data packets** sent and **4 feedback packets** received

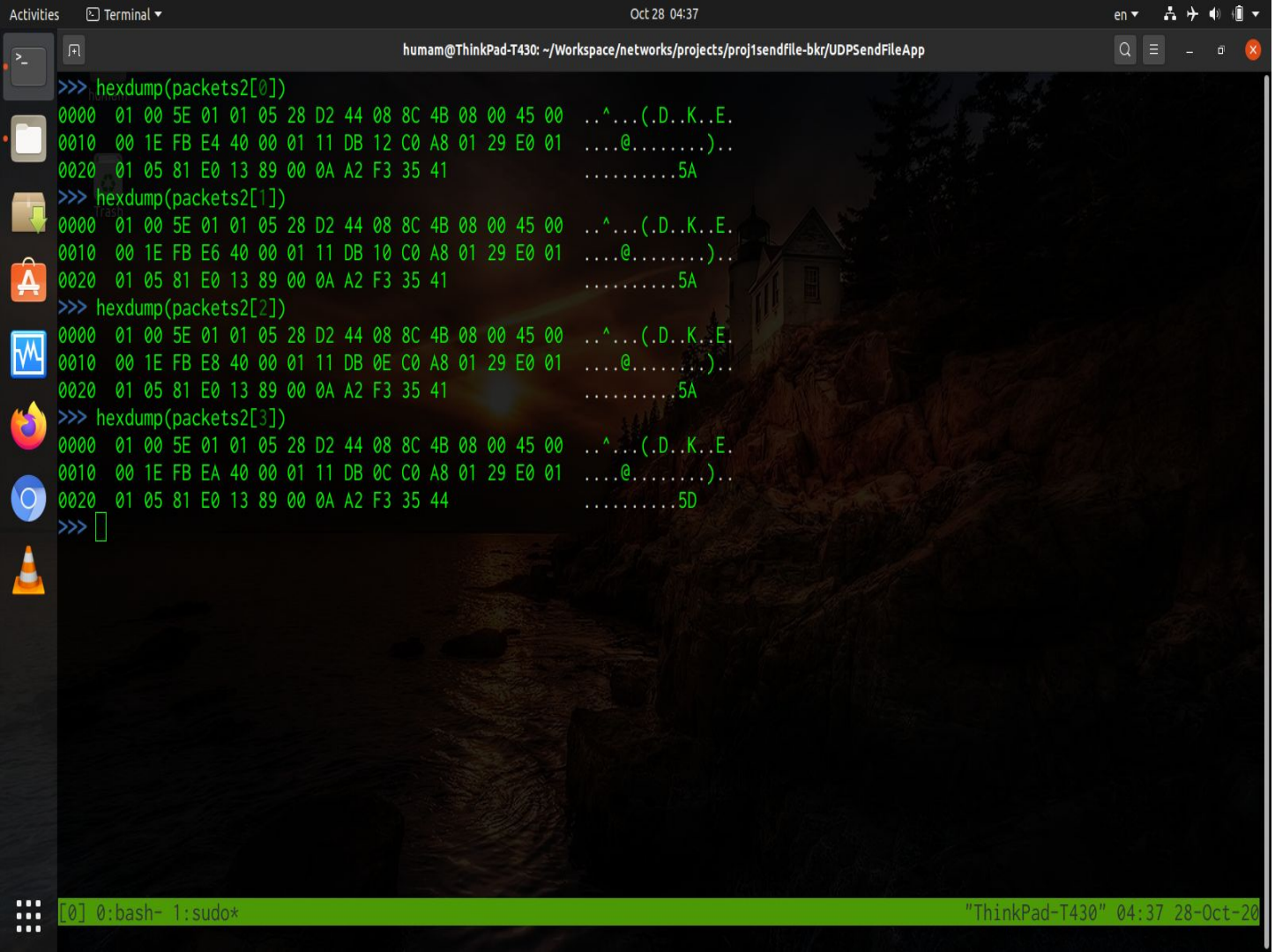


Testing of the **new** application

(manual, **Fail-Fast** version)

File size:
100 bytes

Result:
4 data packets in order are **file name, data size, checksum, file content**

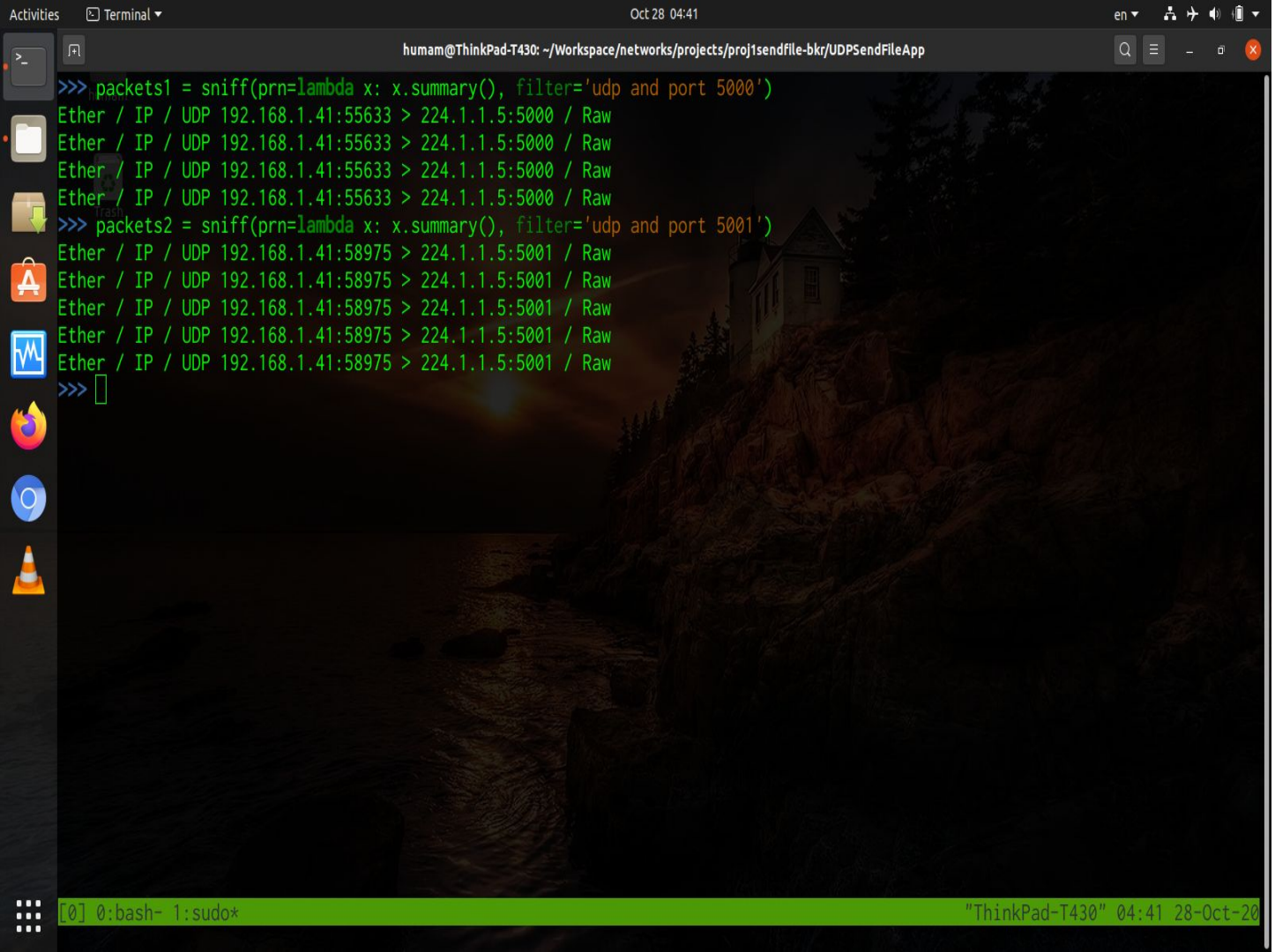


Testing of the **new** application

(manual, **Fail-Fast** version)

File size:
100 bytes

Result:
Feedback packets are **ACKs for metadata and final transmission status confirmation**

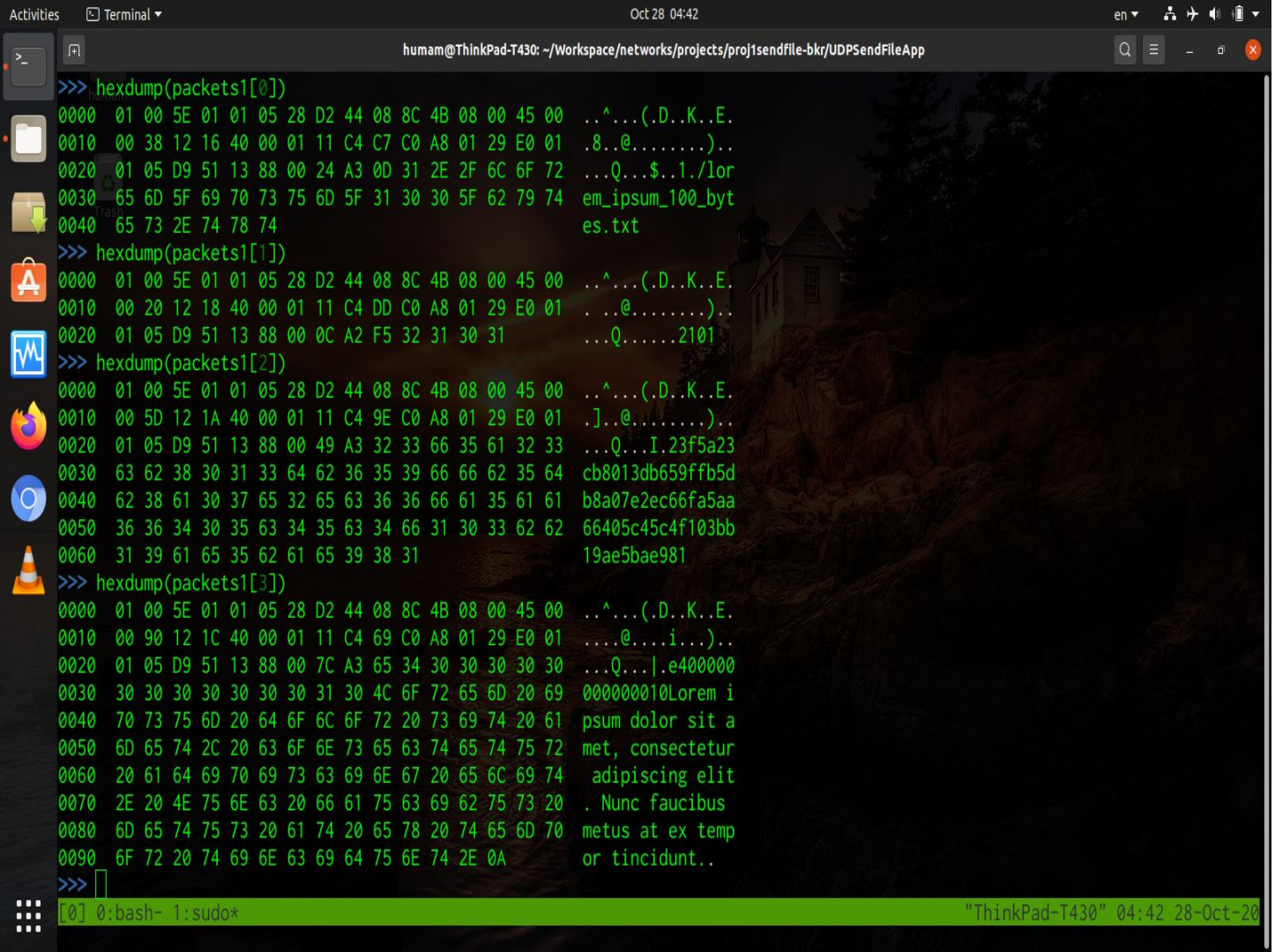


Testing of the **new** application

(manual, **ARQ** version)

File size:
100 bytes

Result:
ScaPy detects **4 data packets** sent and **5 feedback packets** received

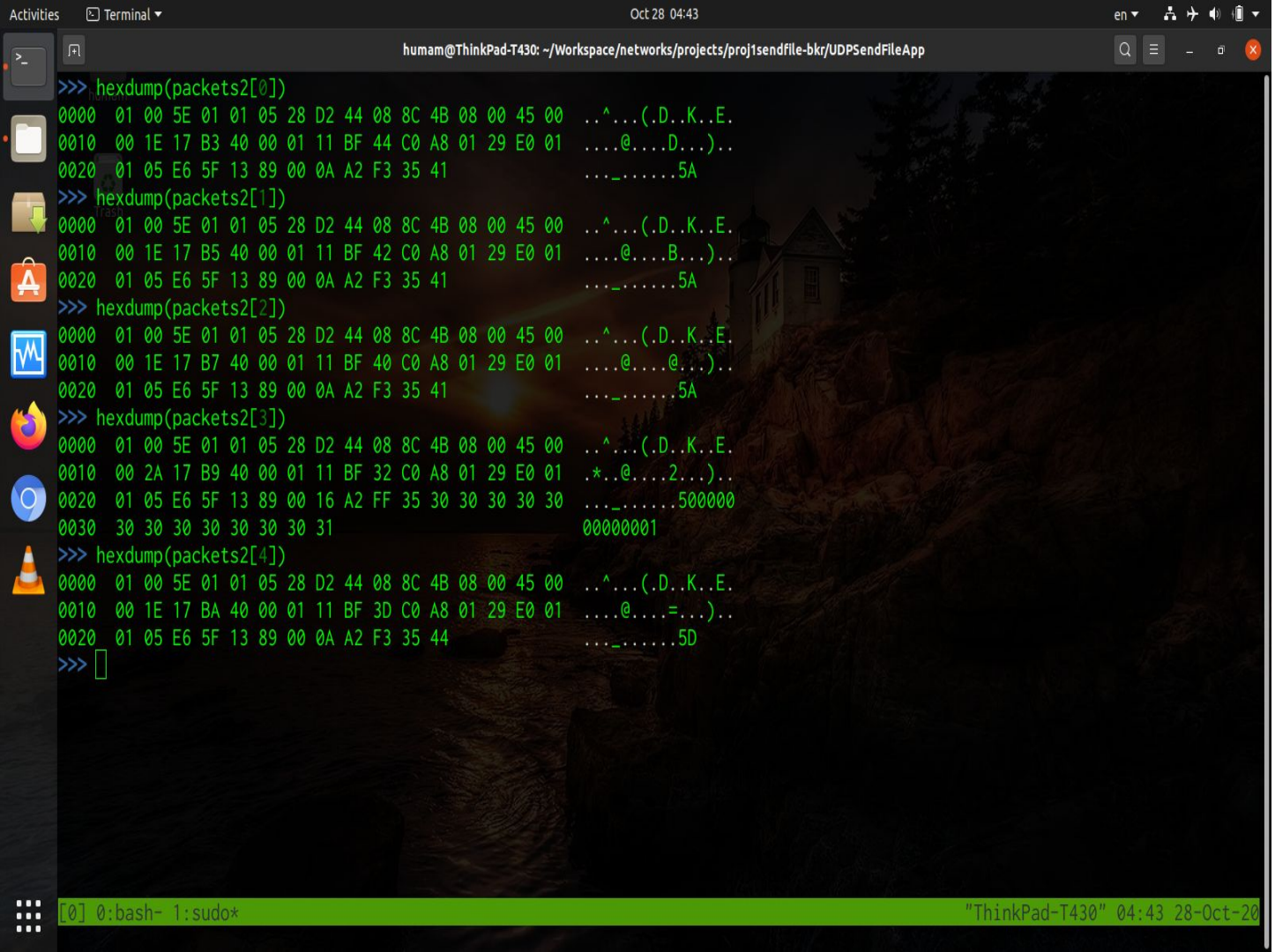


Testing of the **new** application

(manual, **ARQ** version)

File size:
100 bytes

Result:
4 data packets in order are **file name, data size, checksum, file content**



Testing of the **new** application

(manual, **ARQ** version)

File size:
100 bytes

Result:
Feedback packets are **ACKs for metadata, file segment repeat feedback and final transmission status confirmation**

Results Overview

Simple App. vs. UDP `SendFile`

- **Marked improvement in reliability**: zero errors in new application for all practical tests
- **Usability concerns addressed**: clear protocol design and clean implementations to demonstrate practice
- **Anti-corruption policy introduced**: SHA-2 is considered reliable for now
- Remaining issues clarified and demonstrated with examples

Reference

Texts used for research and development

- *Data and Computer Communications* by William Stallings, 10th edition, 2014.
- Email exchange with Prof. Chen
- *UNIX Network Programming* by W. Richard Stevens, 1st edition, 1990.
- RFC 1350: *The TFTP Protocol* (Revision 2) by the Network Working Group, 1992.

The End

...

Presentation by Team BKR

Briskman, Kapialiush, Latif & Rashid