LAB 3: AMQP

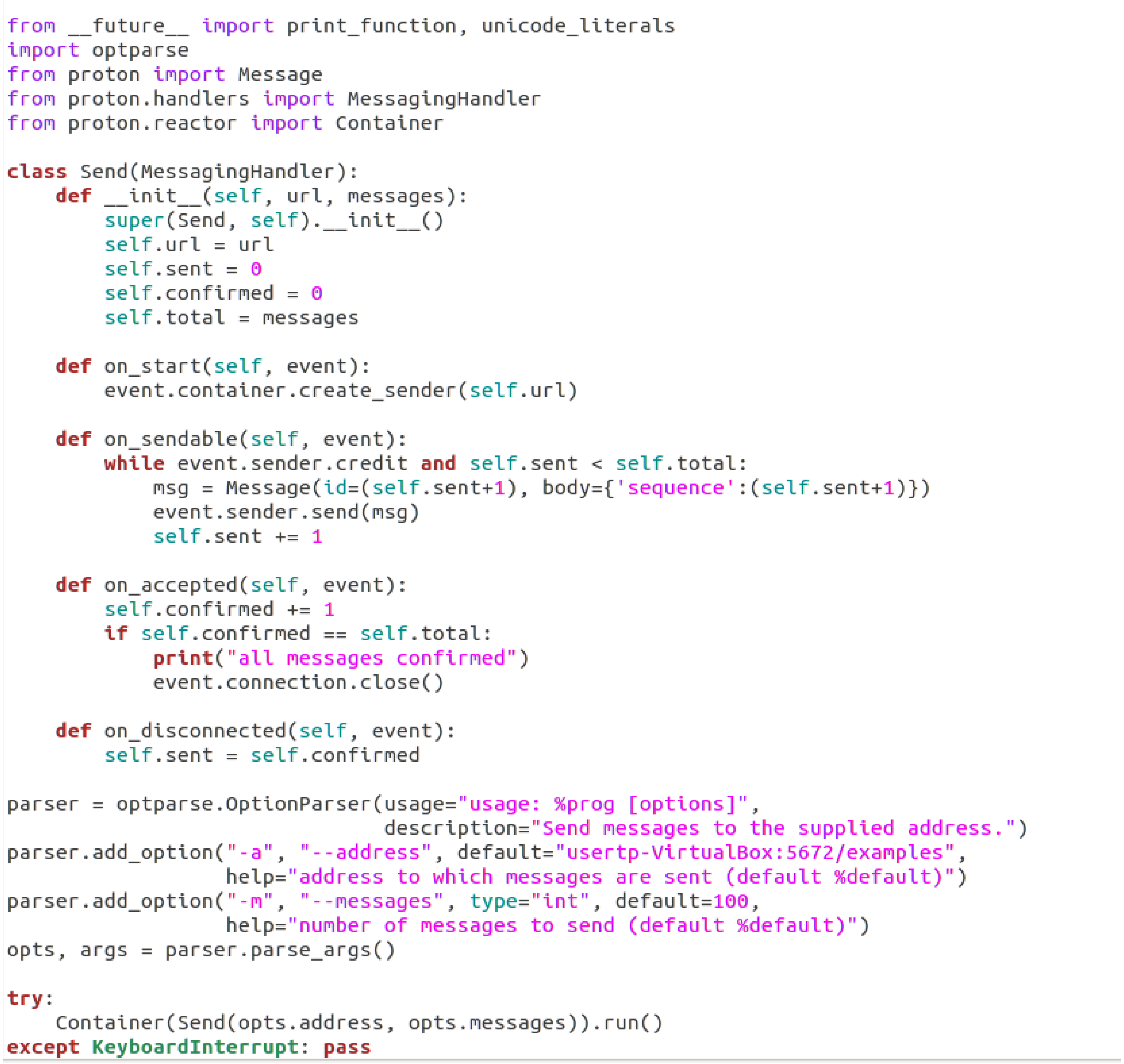
Shubhika GARG

A) PART-1

1. In the directory *"/home/usertp/amqp"*, you may find the code of a sender and a receiver, written in Python.

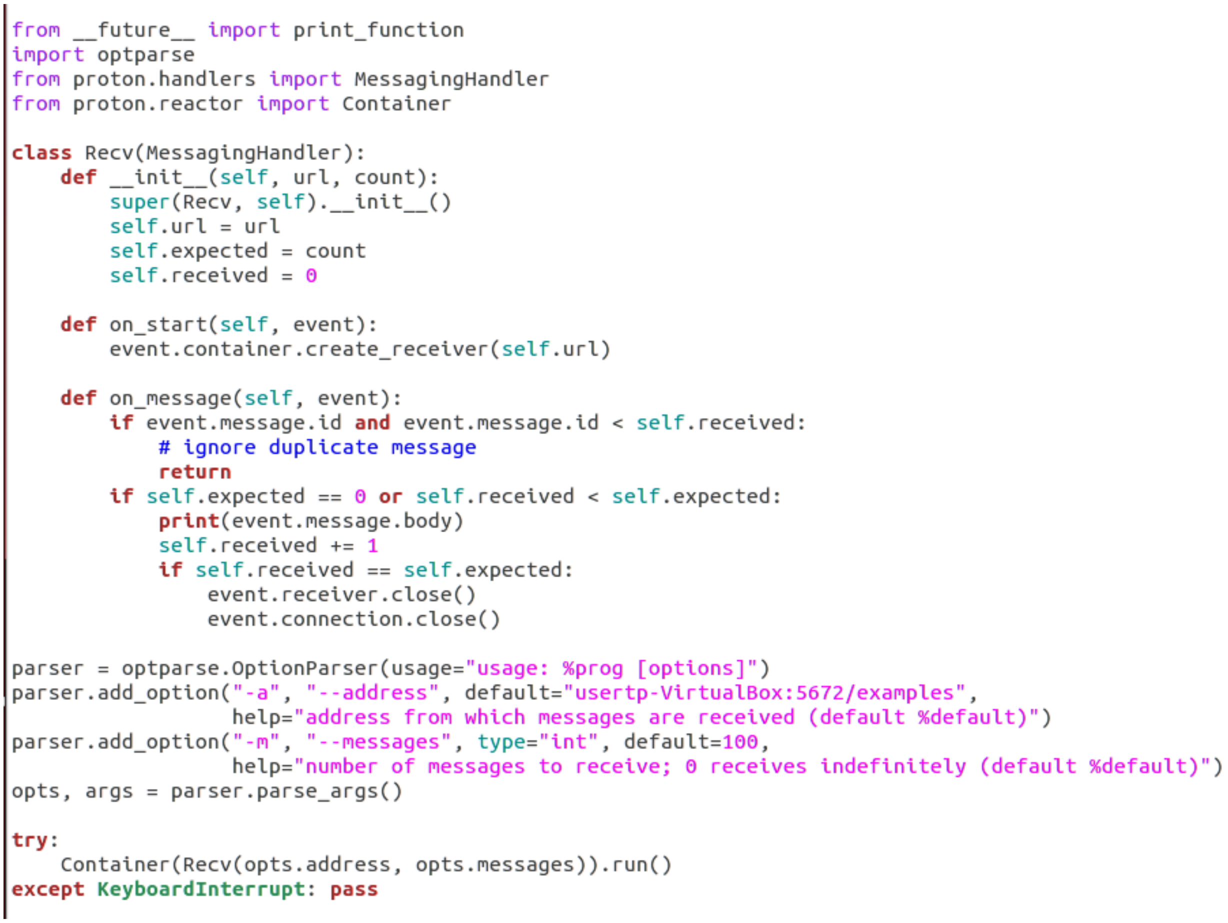
Below are the two python codes:

sender.py



```
from __future__ import print_function, unicode_literals
import optparse
from proton import Message
from proton.handlers import MessagingHandler
from proton.reactor import Container

class Send(MessagingHandler):
    def __init__(self, url, messages):
        super(Send, self).__init__()
        self.url = url
        self.sent = 0
        self.confirmed = 0
        self.total = messages

    def on_start(self, event):
        event.container.create_sender(self.url)

    def on_sendable(self, event):
        while event.sender.credit and self.sent < self.total:
            msg = Message(id=(self.sent+1), body={'sequence':(self.sent+1)})
            event.sender.send(msg)
            self.sent += 1

    def on_accepted(self, event):
        self.confirmed += 1
        if self.confirmed == self.total:
            print("all messages confirmed")
            event.connection.close()

    def on_disconnected(self, event):
        self.sent = self.confirmed

parser = optparse.OptionParser(usage="usage: %prog [options]",
                               description="Send messages to the supplied address.")
parser.add_option("-a", "--address", default="usertp-VirtualBox:5672/examples",
                  help="address to which messages are sent (default %default)")
parser.add_option("-m", "--messages", type="int", default=100,
                  help="number of messages to send (default %default)")
opts, args = parser.parse_args()

try:
    Container(Send(opts.address, opts.messages)).run()
except KeyboardInterrupt: pass
```

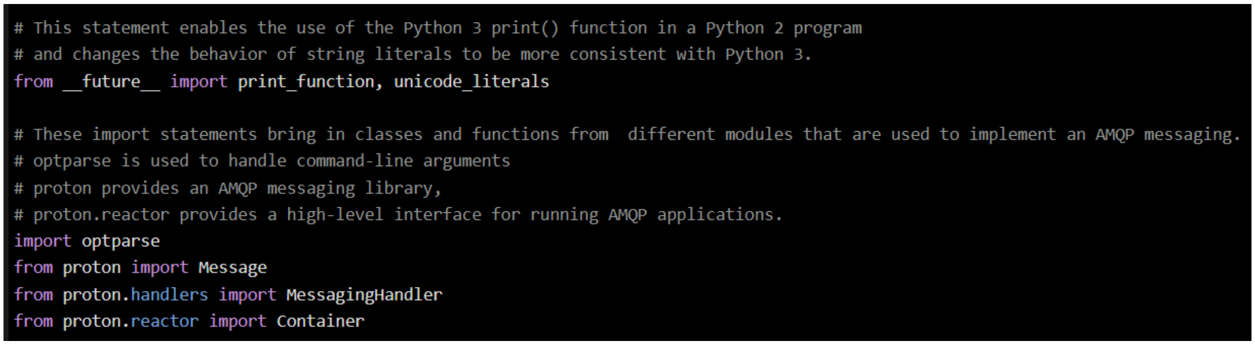
receiver.py



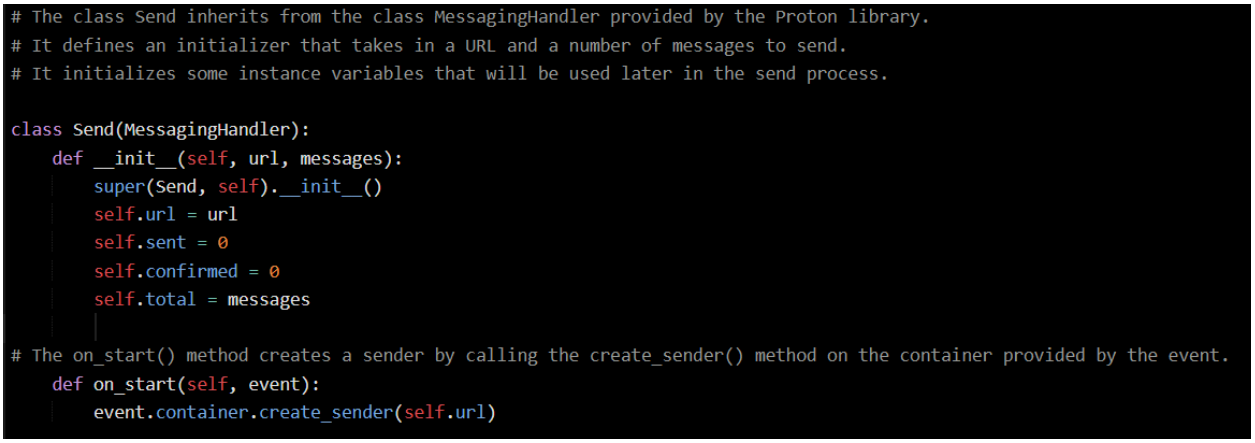
```
from __future__ import print_function
import optparse
from proton.handlers import MessagingHandler
from proton.reactor import Container

class Recv(MessagingHandler):
    def __init__(self, url, count):
        super(Recv, self).__init__()
        self.url = url
        self.expected = count
        self.received = 0

    def on_start(self, event):
        event.container.create_receiver(self.url)

    def on_message(self, event):
        if event.message.id and event.message.id < self.received:
            # ignore duplicate message
            return
        if self.expected == 0 or self.received < self.expected:
            print(event.message.body)
            self.received += 1
            if self.received == self.expected:
                event.receiver.close()
                event.connection.close()

parser = optparse.OptionParser(usage="usage: %prog [options]")
parser.add_option("-a", "--address", default="usertp-VirtualBox:5672/examples",
                  help="address from which messages are received (default %default)")
parser.add_option("-m", "--messages", type="int", default=100,
                  help="number of messages to receive; 0 receives indefinitely (default %default)")
opts, args = parser.parse_args()

try:
    Container(Recv(opts.address, opts.messages)).run()
except KeyboardInterrupt: pass
```

2. Explain each line of the code?

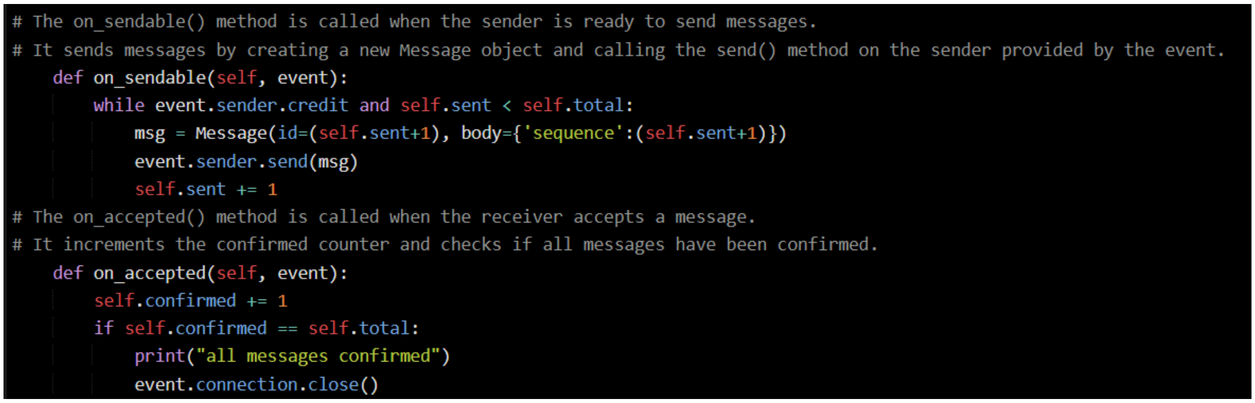
Explanation of the sender code provided in comments:



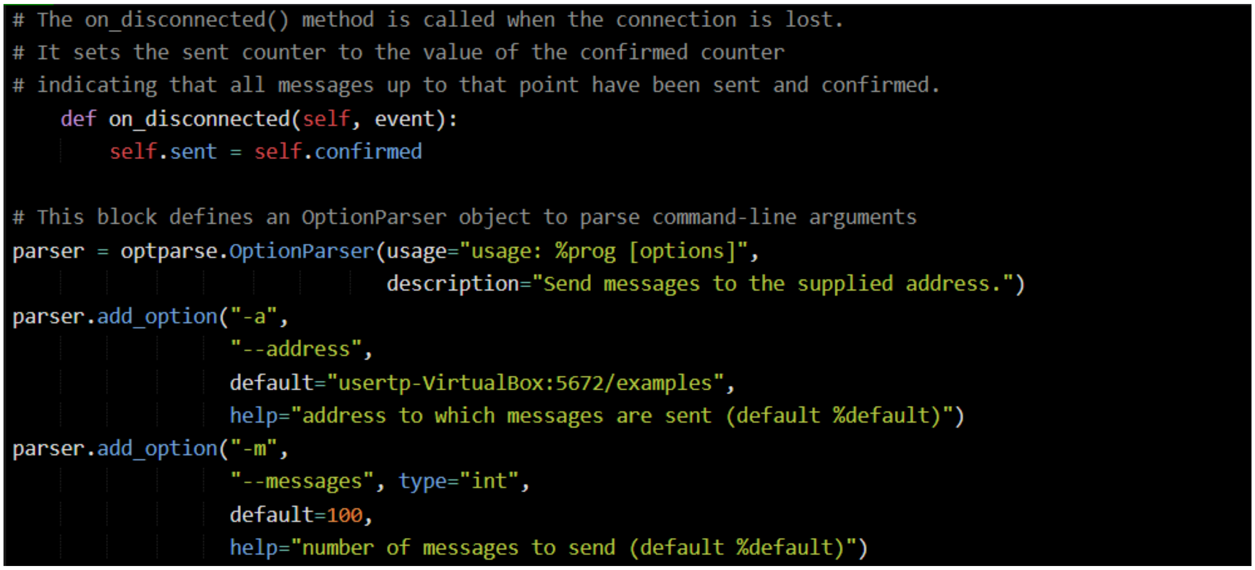
```
# This statement enables the use of the Python 3 print() function in a Python 2 program
# and changes the behavior of string literals to be more consistent with Python 3.
from __future__ import print_function, unicode_literals

# These import statements bring in classes and functions from  different modules that are used to implement an AMQP messaging.
# optparse is used to handle command-line arguments
# proton provides an AMQP messaging library,
# proton.reactor provides a high-level interface for running AMQP applications.
import optparse
from proton import Message
from proton.handlers import MessagingHandler
from proton.reactor import Container
```



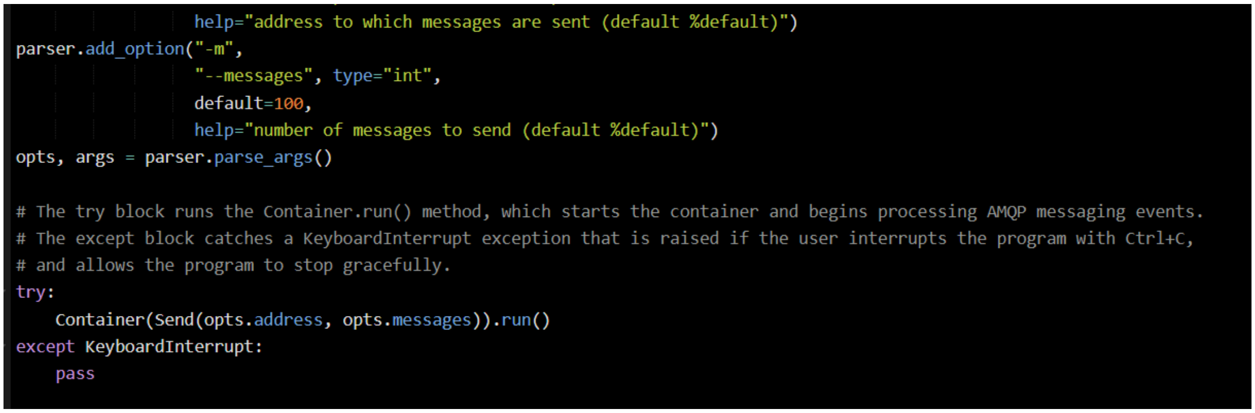
```python
# The class Send inherits from the class MessagingHandler provided by the Proton library.
# It defines an initializer that takes in a URL and a number of messages to send.
# It initializes some instance variables that will be used later in the send process.

class Send(MessagingHandler):
    def __init__(self, url, messages):
        super(Send, self).__init__()
        self.url = url
        self.sent = 0
        self.confirmed = 0
        self.total = messages

# The on_start() method creates a sender by calling the create_sender() method on the container provided by the event.
    def on_start(self, event):
        event.container.create_sender(self.url)
```



```python
# The on_sendable() method is called when the sender is ready to send messages.
# It sends messages by creating a new Message object and calling the send() method on the sender provided by the event.
    def on_sendable(self, event):
        while event.sender.credit and self.sent < self.total:
            msg = Message(id=(self.sent+1), body={'sequence':(self.sent+1)})
            event.sender.send(msg)
            self.sent += 1
# The on_accepted() method is called when the receiver accepts a message.
# It increments the confirmed counter and checks if all messages have been confirmed.
    def on_accepted(self, event):
        self.confirmed += 1
        if self.confirmed == self.total:
            print("all messages confirmed")
            event.connection.close()
```



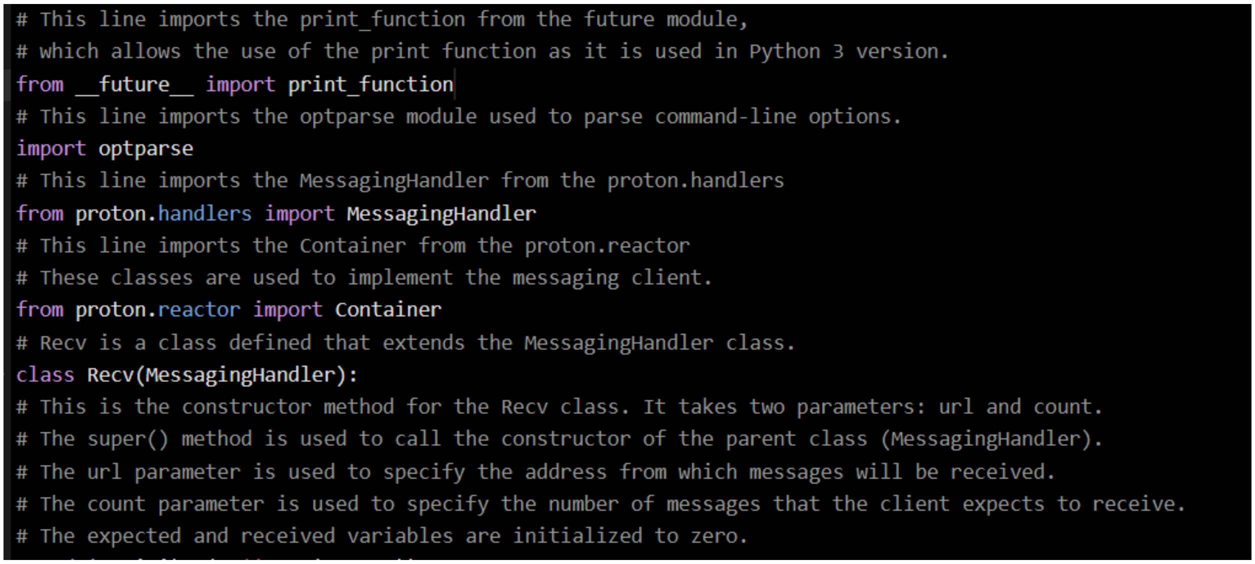
```python
# The on_disconnected() method is called when the connection is lost.
# It sets the sent counter to the value of the confirmed counter
# indicating that all messages up to that point have been sent and confirmed.
    def on_disconnected(self, event):
        self.sent = self.confirmed

# This block defines an OptionParser object to parse command-line arguments
parser = optparse.OptionParser(usage="usage: %prog [options]",
                               description="Send messages to the supplied address.")
parser.add_option("-a",
                  "--address",
                  default="usertp-VirtualBox:5672/examples",
                  help="address to which messages are sent (default %default)")
parser.add_option("-m",
                  "--messages", type="int",
                  default=100,
                  help="number of messages to send (default %default)")
```



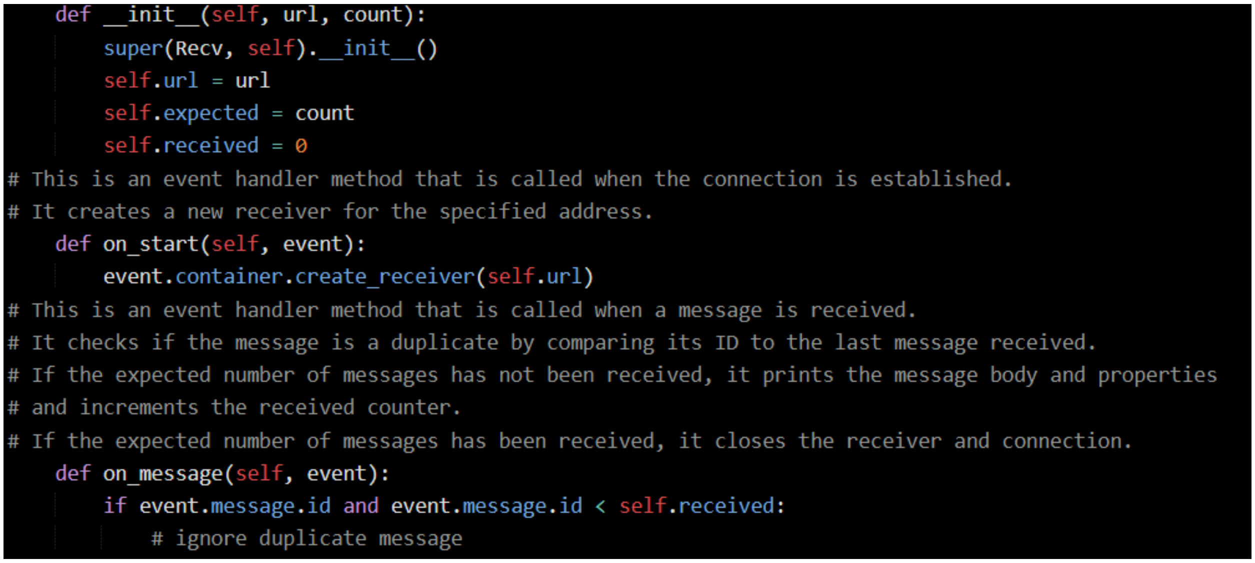
```python
                  help="address to which messages are sent (default %default)")
parser.add_option("-m",
                  "--messages", type="int",
                  default=100,
                  help="number of messages to send (default %default)")
```

```python
opts, args = parser.parse_args()

# The try block runs the Container.run() method, which starts the container and begins processing AMQP messaging events.
# The except block catches a KeyboardInterrupt exception that is raised if the user interrupts the program with Ctrl+C,
# and allows the program to stop gracefully.
try:
    Container(Send(opts.address, opts.messages)).run()
except KeyboardInterrupt:
    pass
```

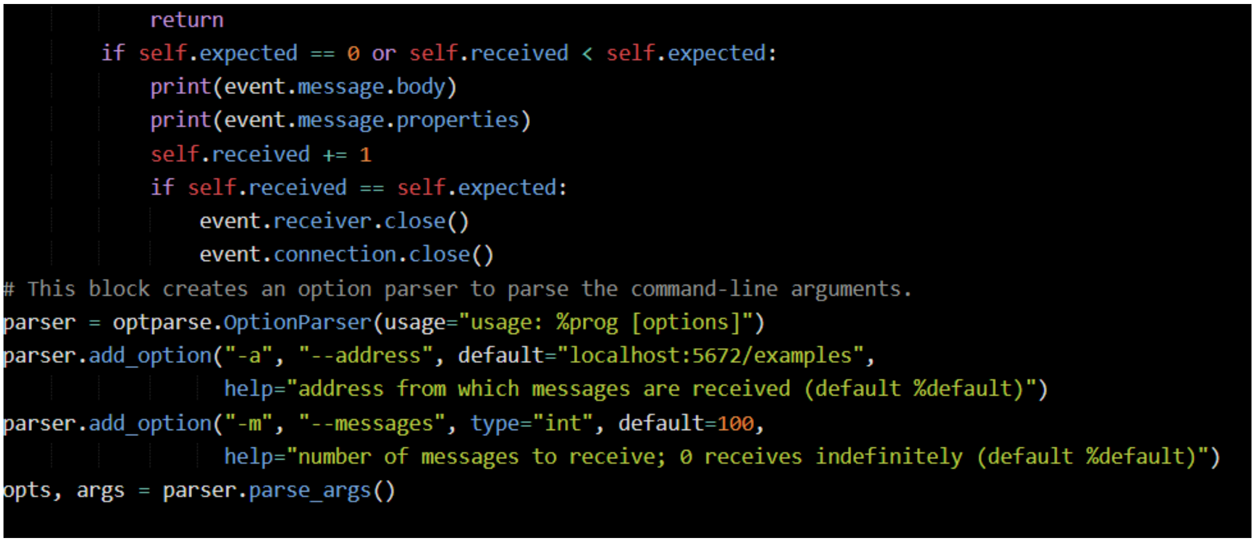
Explanation of receiver code provided in comments:



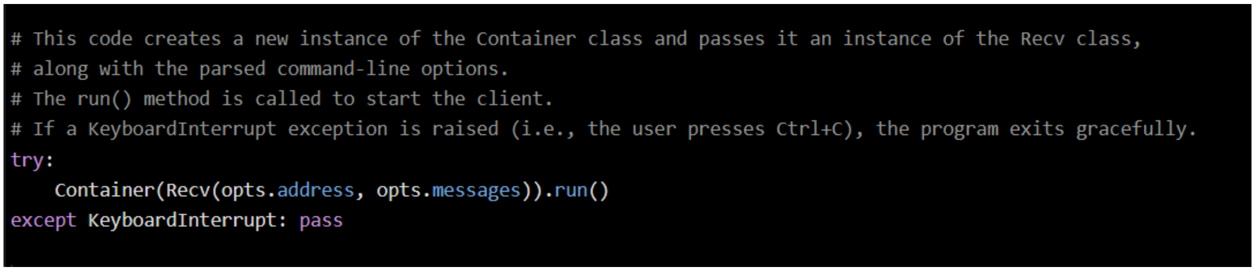
```
# This line imports the print_function from the future module,
# which allows the use of the print function as it is used in Python 3 version.
from __future__ import print_function
# This line imports the optparse module used to parse command-line options.
import optparse
# This line imports the MessagingHandler from the proton.handlers
from proton.handlers import MessagingHandler
# This line imports the Container from the proton.reactor
# These classes are used to implement the messaging client.
from proton.reactor import Container
# Recv is a class defined that extends the MessagingHandler class.
class Recv(MessagingHandler):
# This is the constructor method for the Recv class. It takes two parameters: url and count.
# The super() method is used to call the constructor of the parent class (MessagingHandler).
# The url parameter is used to specify the address from which messages will be received.
# The count parameter is used to specify the number of messages that the client expects to receive.
# The expected and received variables are initialized to zero.
```



```
    def __init__(self, url, count):
        super(Recv, self).__init__()
        self.url = url
        self.expected = count
        self.received = 0
# This is an event handler method that is called when the connection is established.
# It creates a new receiver for the specified address.
    def on_start(self, event):
        event.container.create_receiver(self.url)
# This is an event handler method that is called when a message is received.
# It checks if the message is a duplicate by comparing its ID to the last message received.
# If the expected number of messages has not been received, it prints the message body and properties
# and increments the received counter.
# If the expected number of messages has been received, it closes the receiver and connection.
    def on_message(self, event):
        if event.message.id and event.message.id < self.received:
            # ignore duplicate message
```

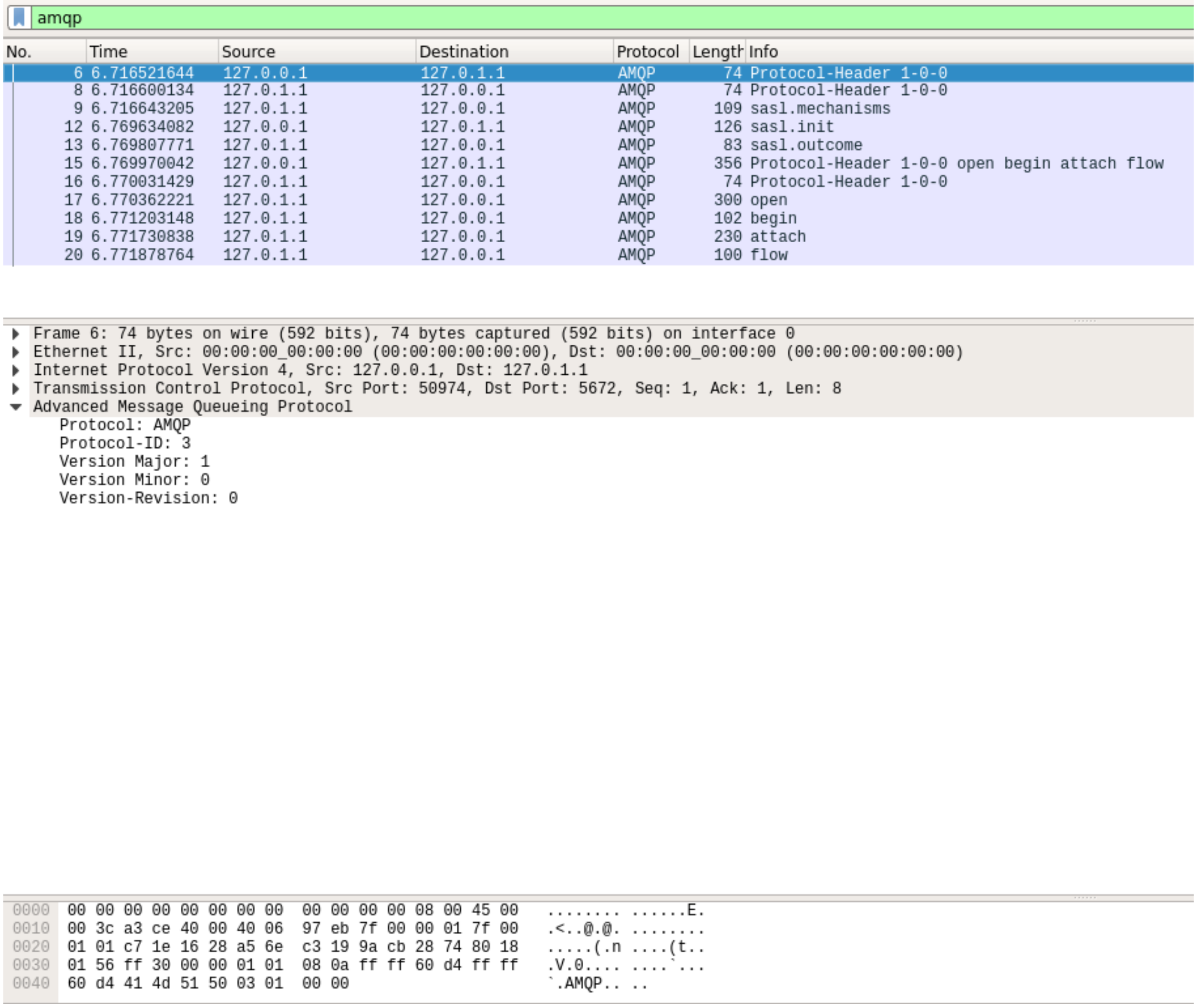


```
            return
        if self.expected == 0 or self.received < self.expected:
            print(event.message.body)
            print(event.message.properties)
            self.received += 1
            if self.received == self.expected:
                event.receiver.close()
                event.connection.close()
# This block creates an option parser to parse the command-line arguments.
parser = optparse.OptionParser(usage="usage: %prog [options]")
parser.add_option("-a", "--address", default="localhost:5672/examples",
                  help="address from which messages are received (default %default)")
parser.add_option("-m", "--messages", type="int", default=100,
                  help="number of messages to receive; 0 receives indefinitely (default %default)")
opts, args = parser.parse_args()
```



```
# This code creates a new instance of the Container class and passes it an instance of the Recv class,
# along with the parsed command-line options.
# The run() method is called to start the client.
# If a KeyboardInterrupt exception is raised (i.e., the user presses Ctrl+C), the program exits gracefully.
try:
    Container(Recv(opts.address, opts.messages)).run()
except KeyboardInterrupt: pass
```

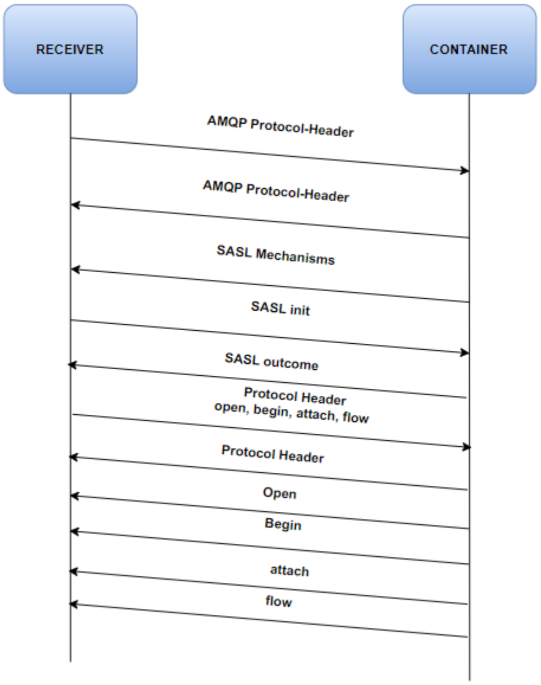
3. Start Wireshark, and launch a capture. You may use AMQP filter to see only AMQP messages. Launch the receiver. Stop the capture.



```
amqp

No.     Time          Source       Destination  Protocol Length Info
      6 6.716521644   127.0.0.1    127.0.1.1    AMQP       74 Protocol-Header 1-0-0
      8 6.716600134   127.0.1.1    127.0.0.1    AMQP       74 Protocol-Header 1-0-0
      9 6.716643205   127.0.1.1    127.0.0.1    AMQP      109 sasl.mechanisms
     12 6.769634082   127.0.0.1    127.0.1.1    AMQP      126 sasl.init
     13 6.769807771   127.0.1.1    127.0.0.1    AMQP       83 sasl.outcome
     15 6.769970042   127.0.0.1    127.0.1.1    AMQP      356 Protocol-Header 1-0-0 open begin attach flow
     16 6.770031429   127.0.1.1    127.0.0.1    AMQP       74 Protocol-Header 1-0-0
     17 6.770362221   127.0.1.1    127.0.0.1    AMQP      300 open
     18 6.771203148   127.0.1.1    127.0.0.1    AMQP      102 begin
     19 6.771730838   127.0.1.1    127.0.0.1    AMQP      230 attach
     20 6.771878764   127.0.1.1    127.0.0.1    AMQP      100 flow

▸ Frame 6: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0
▸ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
▸ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.1.1
▸ Transmission Control Protocol, Src Port: 50974, Dst Port: 5672, Seq: 1, Ack: 1, Len: 8
▾ Advanced Message Queueing Protocol
    Protocol: AMQP
    Protocol-ID: 3
    Version Major: 1
    Version Minor: 0
    Version-Revision: 0

0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ........ ......E.
0010  00 3c a3 ce 40 00 40 06  97 eb 7f 00 00 01 7f 00   .<..@.@. ........
0020  01 01 c7 1e 16 28 a5 6e  c3 19 9a cb 28 74 80 18   .....(.n ....(t..
0030  01 56 ff 30 00 00 01 01  08 0a ff ff 60 d4 ff ff   .V.0.... ....`...
0040  60 d4 41 4d 51 50 03 01  00 00                     `.AMQP.. ..
```

4. What are the exchanged messages? Draw an exchange diagram.

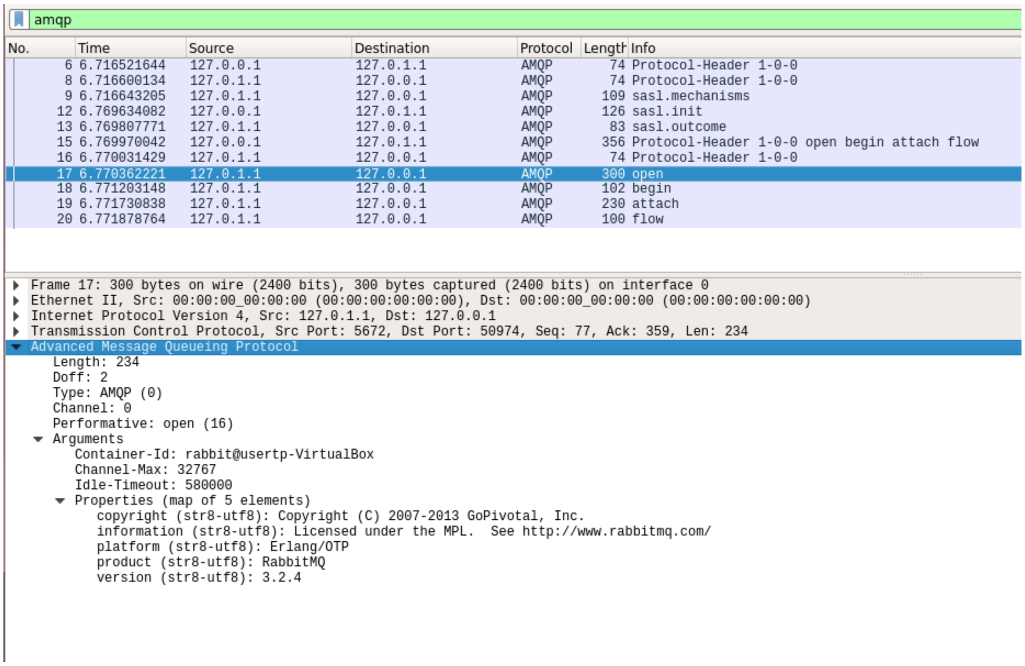
From the above capture, the exchanged messages are Protocol-Header, SASL mechanisms, Open, Begin, Attach and Flow.



5. Describe the characteristics of the connection, session and link? Credit, window size, maximum frame size? What are the objectives behind these values?

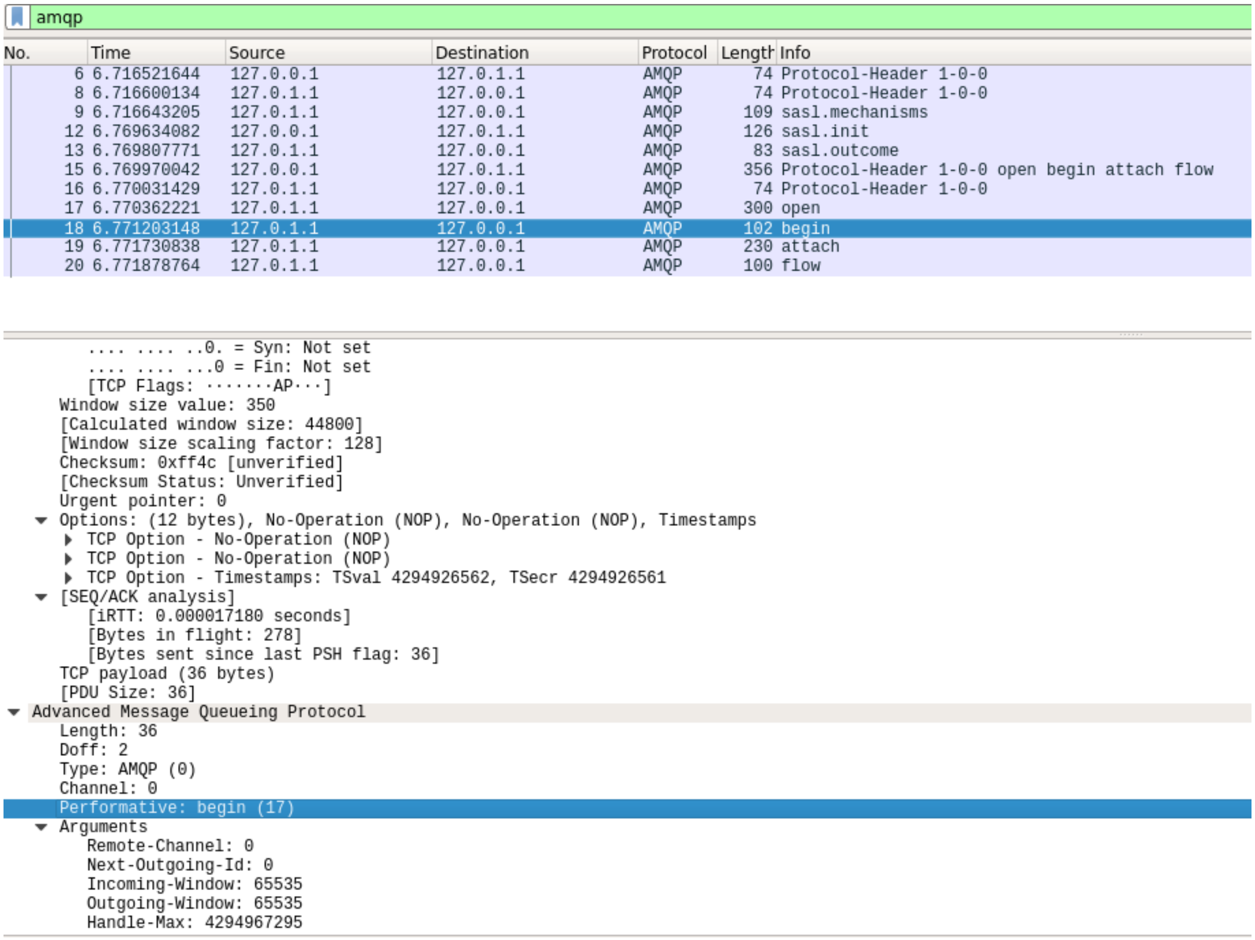
- To establish a connection between two containers in AMQP, the following steps are taken: A TCP connection is established between the two containers. A header frame is exchanged to negotiate the version of AMQP to be used. The Open frame is sent to initiate the connection between the containers. Connections are subject to idle timeout threshold. Timeout is triggered by local peer if no AMQP frames are received after threshold exceeded. The open frame can only be sent on channel 0.
- A session binds together two uni-directional channels to form a bidirectional, sequential conversation between two containers. Session is created using Begin performative
- Links provide a credit-based flow control scheme based on the number of messages transmitted, allowing applications to control which nodes to receive messages from at a given point. Links are created using performative attach.
- The objective of these values is used to control the flow of communication at different layers in the AMQP protocol.

Connection: Container-Id, Channel-Max and Idle Timeout values can be seen in the below capture. Also, the channel below is 0.



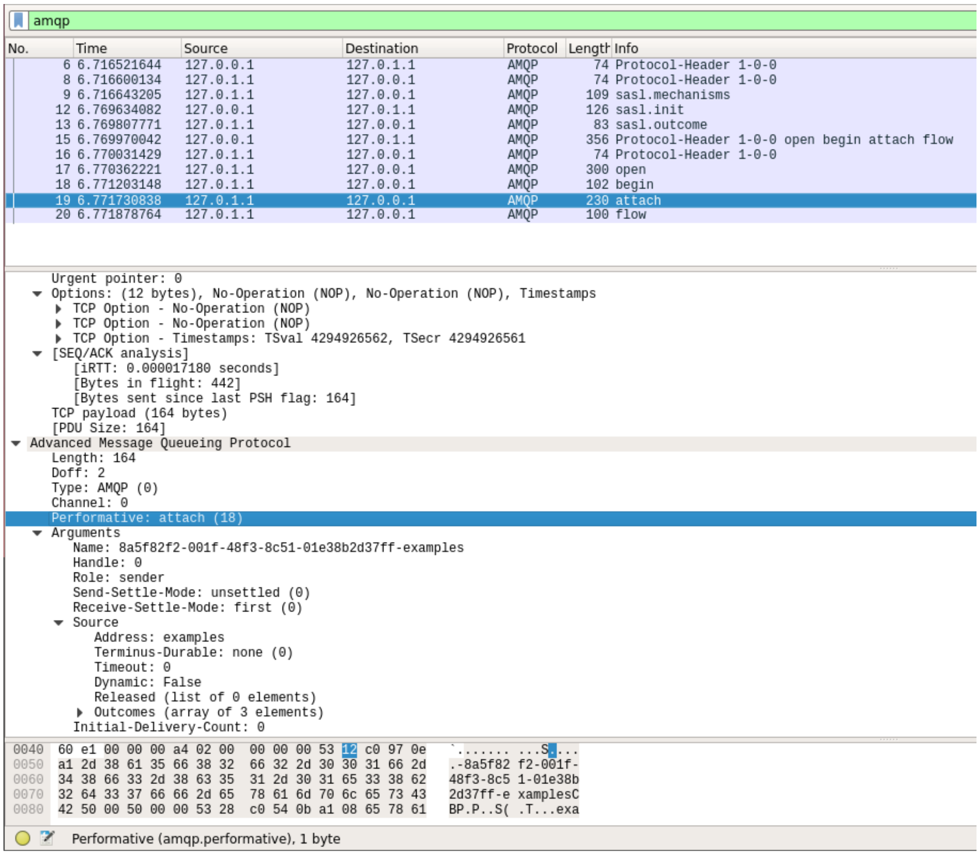
```
amqp

No.    Time          Source       Destination   Protocol  Length Info
     6 6.716521644   127.0.0.1    127.0.1.1     AMQP        74 Protocol-Header 1-0-0
     8 6.716600134   127.0.1.1    127.0.0.1     AMQP        74 Protocol-Header 1-0-0
     9 6.716643205   127.0.1.1    127.0.0.1     AMQP       109 sasl.mechanisms
    12 6.769634082   127.0.0.1    127.0.1.1     AMQP       126 sasl.init
    13 6.769807771   127.0.1.1    127.0.0.1     AMQP        83 sasl.outcome
    15 6.769970042   127.0.0.1    127.0.1.1     AMQP       356 Protocol-Header 1-0-0 open begin attach flow
    16 6.770031429   127.0.1.1    127.0.0.1     AMQP        74 Protocol-Header 1-0-0
    17 6.770362221   127.0.1.1    127.0.0.1     AMQP       300 open
    18 6.771203148   127.0.1.1    127.0.0.1     AMQP       102 begin
    19 6.771730838   127.0.1.1    127.0.0.1     AMQP       230 attach
    20 6.771878764   127.0.1.1    127.0.0.1     AMQP       100 flow

▸ Frame 17: 300 bytes on wire (2400 bits), 300 bytes captured (2400 bits) on interface 0
▸ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
▸ Internet Protocol Version 4, Src: 127.0.1.1, Dst: 127.0.0.1
▸ Transmission Control Protocol, Src Port: 5672, Dst Port: 50974, Seq: 77, Ack: 359, Len: 234
▾ Advanced Message Queueing Protocol
     Length: 234
     Doff: 2
     Type: AMQP (0)
     Channel: 0
     Performative: open (16)
   ▾ Arguments
        Container-Id: rabbit@usertp-VirtualBox
        Channel-Max: 32767
        Idle-Timeout: 580000
      ▾ Properties (map of 5 elements)
           copyright (str8-utf8): Copyright (C) 2007-2013 GoPivotal, Inc.
           information (str8-utf8): Licensed under the MPL.  See http://www.rabbitmq.com/
           platform (str8-utf8): Erlang/OTP
           product (str8-utf8): RabbitMQ
           version (str8-utf8): 3.2.4
```

Session: Incoming-Window, Outgoing-Window and Handle-Max values can be seen in the below capture.



```
amqp

No.    Time          Source       Destination   Protocol  Length Info
     6 6.716521644   127.0.0.1    127.0.1.1     AMQP        74 Protocol-Header 1-0-0
     8 6.716600134   127.0.1.1    127.0.0.1     AMQP        74 Protocol-Header 1-0-0
     9 6.716643205   127.0.1.1    127.0.0.1     AMQP       109 sasl.mechanisms
    12 6.769634082   127.0.0.1    127.0.1.1     AMQP       126 sasl.init
    13 6.769807771   127.0.1.1    127.0.0.1     AMQP        83 sasl.outcome
    15 6.769970042   127.0.0.1    127.0.1.1     AMQP       356 Protocol-Header 1-0-0 open begin attach flow
    16 6.770031429   127.0.1.1    127.0.0.1     AMQP        74 Protocol-Header 1-0-0
    17 6.770362221   127.0.1.1    127.0.0.1     AMQP       300 open
    18 6.771203148   127.0.1.1    127.0.0.1     AMQP       102 begin
    19 6.771730838   127.0.1.1    127.0.0.1     AMQP       230 attach
    20 6.771878764   127.0.1.1    127.0.0.1     AMQP       100 flow

        .... .... ..0. = Syn: Not set
        .... .... ...0 = Fin: Not set
        [TCP Flags: ·······AP···]
     Window size value: 350
     [Calculated window size: 44800]
     [Window size scaling factor: 128]
     Checksum: 0xff4c [unverified]
     [Checksum Status: Unverified]
     Urgent pointer: 0
   ▾ Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
      ▸ TCP Option - No-Operation (NOP)
      ▸ TCP Option - No-Operation (NOP)
      ▸ TCP Option - Timestamps: TSval 4294926562, TSecr 4294926561
   ▾ [SEQ/ACK analysis]
        [iRTT: 0.000017180 seconds]
        [Bytes in flight: 278]
        [Bytes sent since last PSH flag: 36]
     TCP payload (36 bytes)
     [PDU Size: 36]
▾ Advanced Message Queueing Protocol
     Length: 36
     Doff: 2
     Type: AMQP (0)
     Channel: 0
     Performative: begin (17)
   ▾ Arguments
        Remote-Channel: 0
        Next-Outgoing-Id: 0
        Incoming-Window: 65535
        Outgoing-Window: 65535
        Handle-Max: 4294967295
```

Link: Link-credit, target and address values can be seen in the below capture.



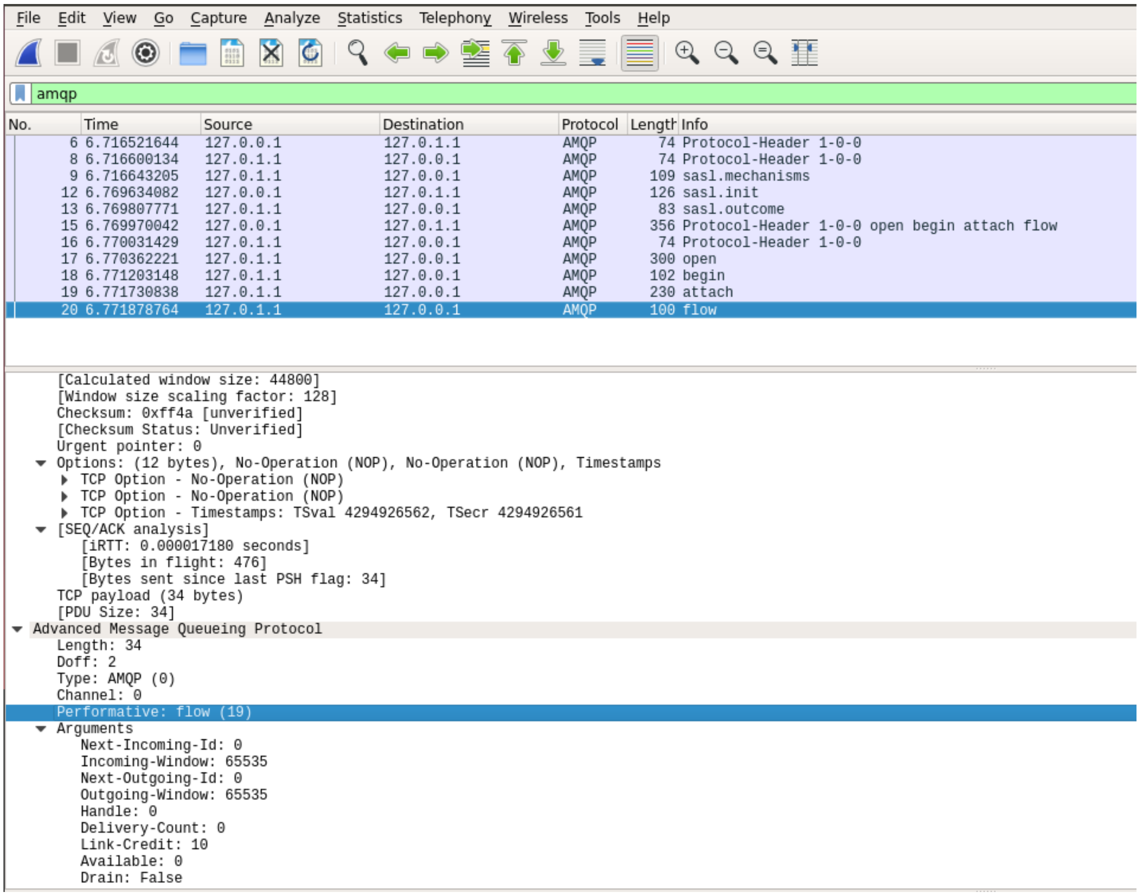
amqp

```
No.    Time          Source      Destination  Protocol Length Info
     6 6.716521644   127.0.0.1   127.0.1.1    AMQP      74 Protocol-Header 1-0-0
     8 6.716600134   127.0.1.1   127.0.0.1    AMQP      74 Protocol-Header 1-0-0
     9 6.716643205   127.0.1.1   127.0.0.1    AMQP     109 sasl.mechanisms
    12 6.769634082   127.0.0.1   127.0.1.1    AMQP     126 sasl.init
    13 6.769807771   127.0.1.1   127.0.0.1    AMQP      83 sasl.outcome
    15 6.769970042   127.0.0.1   127.0.1.1    AMQP     356 Protocol-Header 1-0-0 open begin attach flow
    16 6.770031429   127.0.1.1   127.0.0.1    AMQP      74 Protocol-Header 1-0-0
    17 6.770362221   127.0.1.1   127.0.0.1    AMQP     300 open
    18 6.771203148   127.0.1.1   127.0.0.1    AMQP     102 begin
    19 6.771730838   127.0.1.1   127.0.0.1    AMQP     230 attach
    20 6.771878764   127.0.1.1   127.0.0.1    AMQP     100 flow
```

```
    Urgent pointer: 0
  ▾ Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
    ▸ TCP Option - No-Operation (NOP)
    ▸ TCP Option - No-Operation (NOP)
    ▸ TCP Option - Timestamps: TSval 4294926562, TSecr 4294926561
  ▾ [SEQ/ACK analysis]
      [iRTT: 0.000017180 seconds]
      [Bytes in flight: 442]
      [Bytes sent since last PSH flag: 164]
    TCP payload (164 bytes)
    [PDU Size: 164]
▾ Advanced Message Queueing Protocol
    Length: 164
    Doff: 2
    Type: AMQP (0)
    Channel: 0
    Performative: attach (18)
  ▾ Arguments
      Name: 8a5f82f2-001f-48f3-8c51-01e38b2d37ff-examples
      Handle: 0
      Role: sender
      Send-Settle-Mode: unsettled (0)
      Receive-Settle-Mode: first (0)
    ▾ Source
        Address: examples
        Terminus-Durable: none (0)
        Timeout: 0
        Dynamic: False
        Released (list of 0 elements)
      ▸ Outcomes (array of 3 elements)
      Initial-Delivery-Count: 0
```

```
0040  60 e1 00 00 00 a4 02 00  00 00 00 53 12 c0 97 0e   `....... ...S....
0050  a1 2d 38 61 35 66 38 32  66 32 2d 30 30 31 66 2d   .-8a5f82 f2-001f-
0060  34 38 66 33 2d 38 63 35  31 2d 30 31 65 33 38 62   48f3-8c5 1-01e38b
0070  32 64 33 37 66 66 2d 65  78 61 6d 70 6c 65 73 43   2d37ff-e xamplesC
0080  42 50 00 50 00 00 53 28  c0 54 0b a1 08 65 78 61   BP.P..S( .T...exa
```

Performative (amqp.performative), 1 byte



amqp

```
No.    Time          Source      Destination  Protocol Length Info
     6 6.716521644   127.0.0.1   127.0.1.1    AMQP      74 Protocol-Header 1-0-0
     8 6.716600134   127.0.1.1   127.0.0.1    AMQP      74 Protocol-Header 1-0-0
     9 6.716643205   127.0.1.1   127.0.0.1    AMQP     109 sasl.mechanisms
    12 6.769634082   127.0.0.1   127.0.1.1    AMQP     126 sasl.init
    13 6.769807771   127.0.1.1   127.0.0.1    AMQP      83 sasl.outcome
    15 6.769970042   127.0.0.1   127.0.1.1    AMQP     356 Protocol-Header 1-0-0 open begin attach flow
    16 6.770031429   127.0.1.1   127.0.0.1    AMQP      74 Protocol-Header 1-0-0
    17 6.770362221   127.0.1.1   127.0.0.1    AMQP     300 open
    18 6.771203148   127.0.1.1   127.0.0.1    AMQP     102 begin
    19 6.771730838   127.0.1.1   127.0.0.1    AMQP     230 attach
    20 6.771878764   127.0.1.1   127.0.0.1    AMQP     100 flow
```

```
    [Calculated window size: 44800]
    [Window size scaling factor: 128]
    Checksum: 0xff4a [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
  ▾ Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
    ▸ TCP Option - No-Operation (NOP)
    ▸ TCP Option - No-Operation (NOP)
    ▸ TCP Option - Timestamps: TSval 4294926562, TSecr 4294926561
  ▾ [SEQ/ACK analysis]
      [iRTT: 0.000017180 seconds]
      [Bytes in flight: 476]
      [Bytes sent since last PSH flag: 34]
    TCP payload (34 bytes)
    [PDU Size: 34]
▾ Advanced Message Queueing Protocol
    Length: 34
    Doff: 2
    Type: AMQP (0)
    Channel: 0
    Performative: flow (19)
  ▾ Arguments
      Next-Incoming-Id: 0
      Incoming-Window: 65535
      Next-Outgoing-Id: 0
      Outgoing-Window: 65535
      Handle: 0
      Delivery-Count: 0
      Link-Credit: 10
      Available: 0
      Drain: False
```

6. What are the used performatives? At which level is this information?

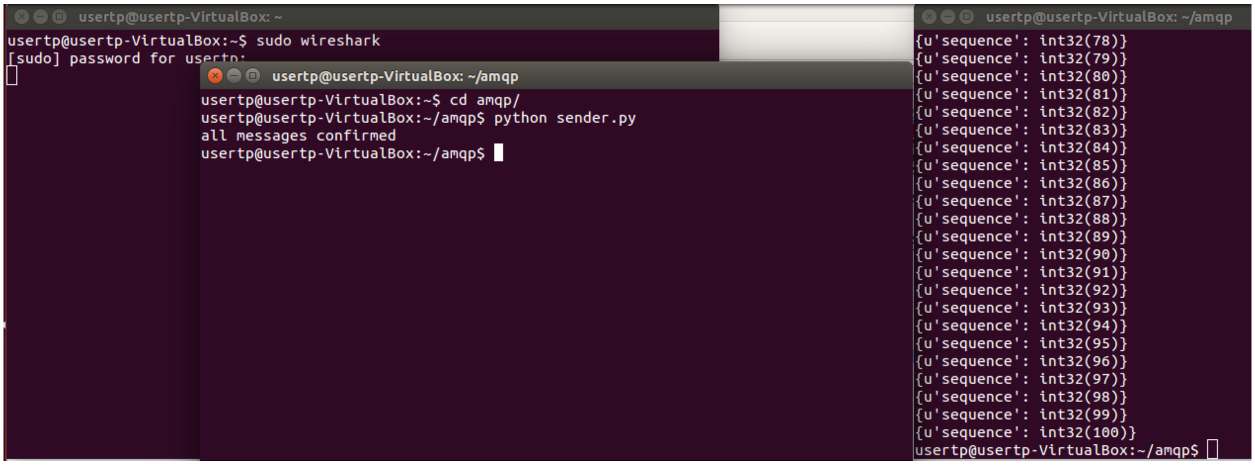
- **Open**: Connection level
- **Begin**: Session level
- **Attach**: Link level
- **Flow**: Link level

7. What is the "id" of the used channel?

The id of the used channel is **0**.

8. Start Wireshark, and launch a capture. Launch the sender. Stop the capture.

Terminal Captures on launching the sender:

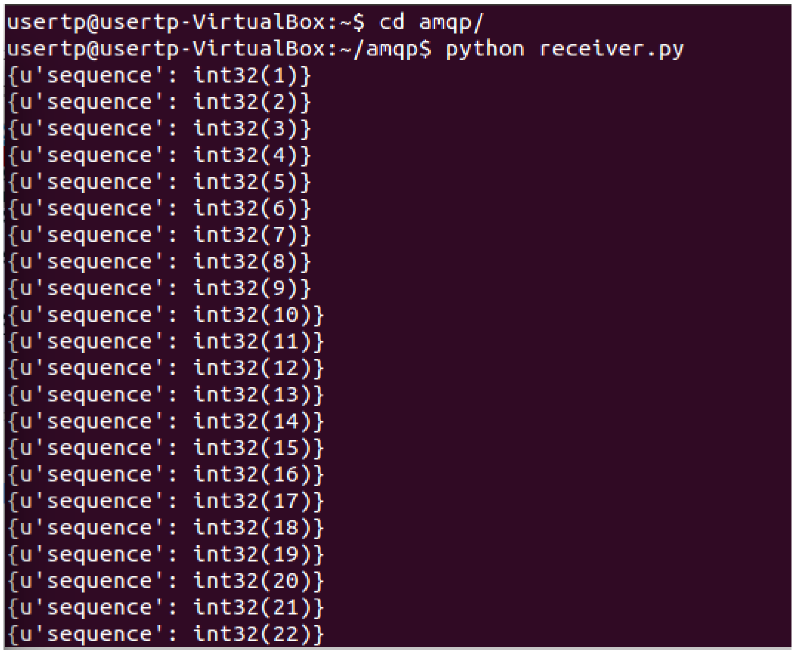


usertp@usertp-VirtualBox: ~
usertp@usertp-VirtualBox: ~/amqp
usertp@usertp-VirtualBox: ~/amqp

```
usertp@usertp-VirtualBox:~$ sudo wireshark
[sudo] password for usertp:
```

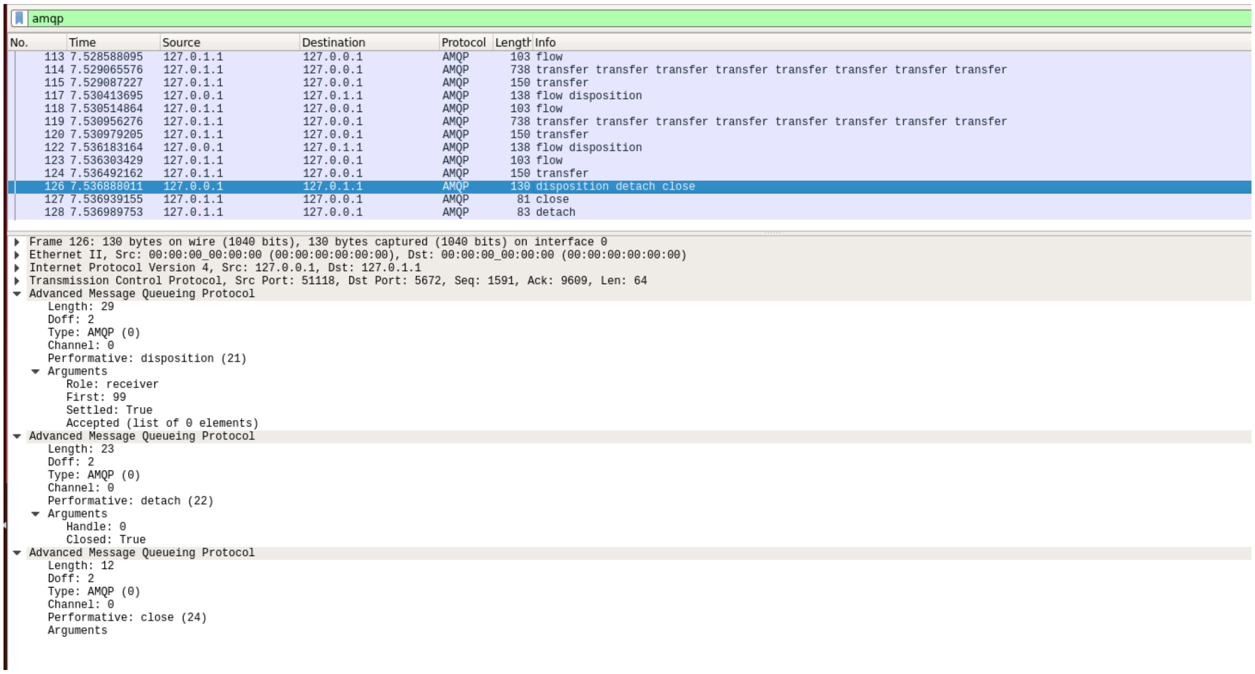
```
usertp@usertp-VirtualBox:~$ cd amqp/
usertp@usertp-VirtualBox:~/amqp$ python sender.py
all messages confirmed
usertp@usertp-VirtualBox:~/amqp$
```

```
{u'sequence': int32(78)}
{u'sequence': int32(79)}
{u'sequence': int32(80)}
{u'sequence': int32(81)}
{u'sequence': int32(82)}
{u'sequence': int32(83)}
{u'sequence': int32(84)}
{u'sequence': int32(85)}
{u'sequence': int32(86)}
{u'sequence': int32(87)}
{u'sequence': int32(88)}
{u'sequence': int32(89)}
{u'sequence': int32(90)}
{u'sequence': int32(91)}
{u'sequence': int32(92)}
{u'sequence': int32(93)}
{u'sequence': int32(94)}
{u'sequence': int32(95)}
{u'sequence': int32(96)}
{u'sequence': int32(97)}
{u'sequence': int32(98)}
{u'sequence': int32(99)}
{u'sequence': int32(100)}
usertp@usertp-VirtualBox:~/amqp$
```



```
usertp@usertp-VirtualBox:~$ cd amqp/
usertp@usertp-VirtualBox:~/amqp$ python receiver.py
{u'sequence': int32(1)}
{u'sequence': int32(2)}
{u'sequence': int32(3)}
{u'sequence': int32(4)}
{u'sequence': int32(5)}
{u'sequence': int32(6)}
{u'sequence': int32(7)}
{u'sequence': int32(8)}
{u'sequence': int32(9)}
{u'sequence': int32(10)}
{u'sequence': int32(11)}
{u'sequence': int32(12)}
{u'sequence': int32(13)}
{u'sequence': int32(14)}
{u'sequence': int32(15)}
{u'sequence': int32(16)}
{u'sequence': int32(17)}
{u'sequence': int32(18)}
{u'sequence': int32(19)}
{u'sequence': int32(20)}
{u'sequence': int32(21)}
{u'sequence': int32(22)}
```

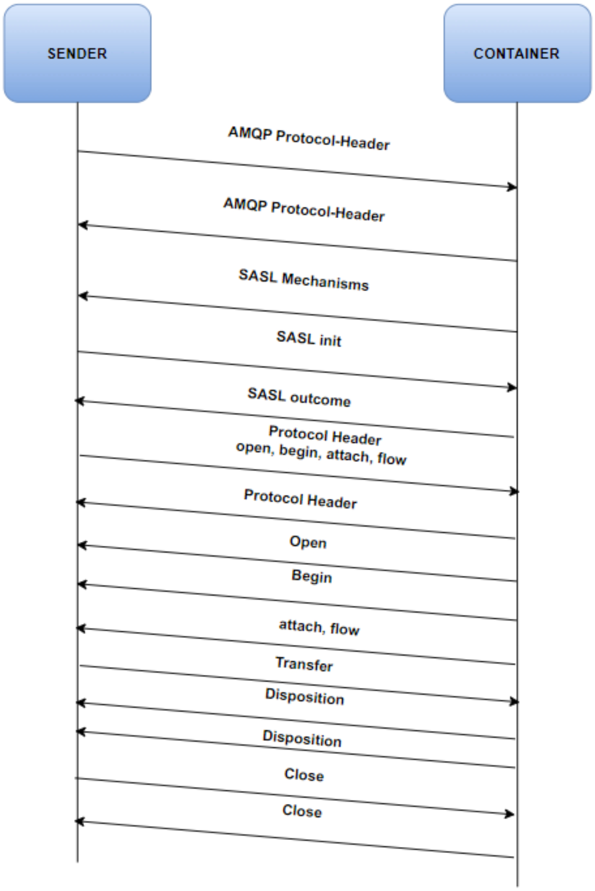
Below is a Wireshark capture:



```
amqp
No.   Time         Source      Destination  Protocol Length Info
  113 7.528588095  127.0.1.1   127.0.0.1    AMQP      103 flow
  114 7.529065576  127.0.1.1   127.0.0.1    AMQP      738 transfer transfer transfer transfer transfer transfer transfer transfer
  115 7.529087227  127.0.1.1   127.0.0.1    AMQP      150 transfer
  117 7.530413695  127.0.0.1   127.0.1.1    AMQP      138 flow disposition
  118 7.530514864  127.0.1.1   127.0.0.1    AMQP      103 flow
  119 7.530956276  127.0.1.1   127.0.0.1    AMQP      738 transfer transfer transfer transfer transfer transfer transfer transfer
  120 7.530979205  127.0.1.1   127.0.0.1    AMQP      150 transfer
  122 7.536183164  127.0.0.1   127.0.1.1    AMQP      138 flow disposition
  123 7.536303429  127.0.1.1   127.0.0.1    AMQP      103 flow
  124 7.536492162  127.0.1.1   127.0.0.1    AMQP      150 transfer
  126 7.536888011  127.0.0.1   127.0.1.1    AMQP      130 disposition detach close
  127 7.536939155  127.0.1.1   127.0.0.1    AMQP       81 close
  128 7.536989753  127.0.1.1   127.0.0.1    AMQP       83 detach

▸ Frame 126: 130 bytes on wire (1040 bits), 130 bytes captured (1040 bits) on interface 0
▸ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
▸ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.1.1
▸ Transmission Control Protocol, Src Port: 51118, Dst Port: 5672, Seq: 1591, Ack: 9609, Len: 64
▾ Advanced Message Queueing Protocol
    Length: 29
    Doff: 2
    Type: AMQP (0)
    Channel: 0
    Performative: disposition (21)
  ▾ Arguments
      Role: receiver
      First: 99
      Settled: True
      Accepted (list of 0 elements)
▾ Advanced Message Queueing Protocol
    Length: 23
    Doff: 2
    Type: AMQP (0)
    Channel: 0
    Performative: detach (22)
  ▾ Arguments
      Handle: 0
      Closed: True
▾ Advanced Message Queueing Protocol
    Length: 12
    Doff: 2
    Type: AMQP (0)
    Channel: 0
    Performative: close (24)
    Arguments
```

9. What are the exchanged messages? Draw an exchange diagram.

The exchanged messages are Protocol-Header, SASL mechanisms, open, begin, attach, flow, transfer, deposition and close.

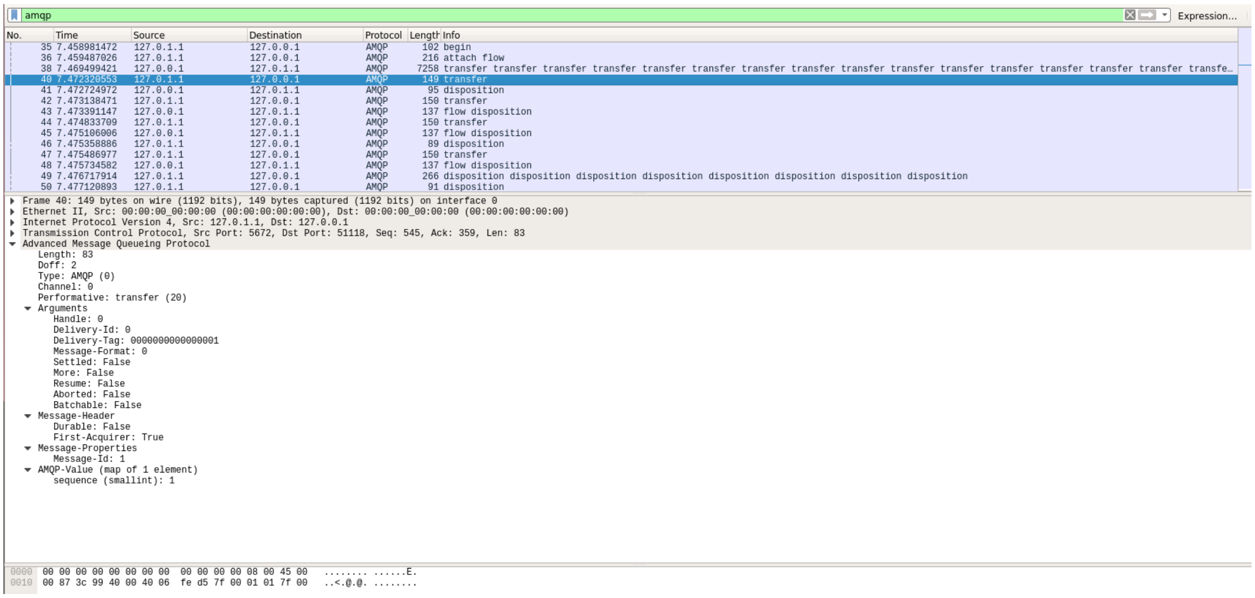


10. What are the used performatives? Explain?

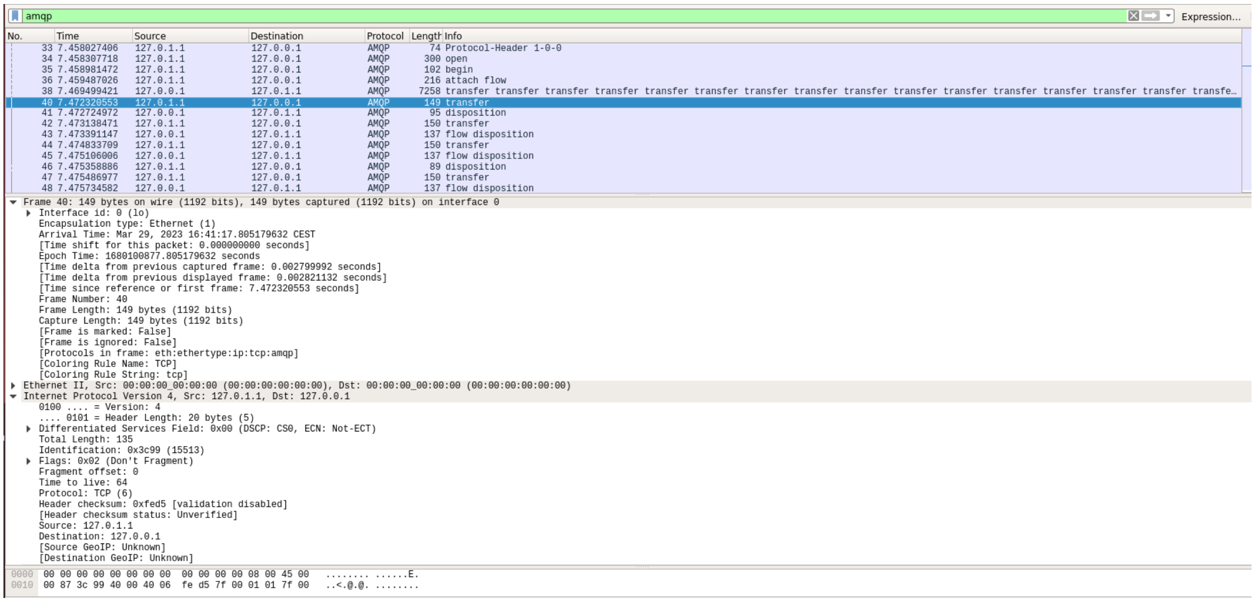
- Open - To open a connection.
- Begin - To begin a session.
- Attach - To attach a link to a session.
- Flow - To control the flow of messages on a link.
- Transfer - To send messages on a link.
- Disposition - To inform the receiver of the outcome of message processing.
- Close - To close a connection or a session.

11. Locate in the "transfer" packet the frame header and the message header? How many messages are present? Why?

Below captures are for the "transfer" packet:

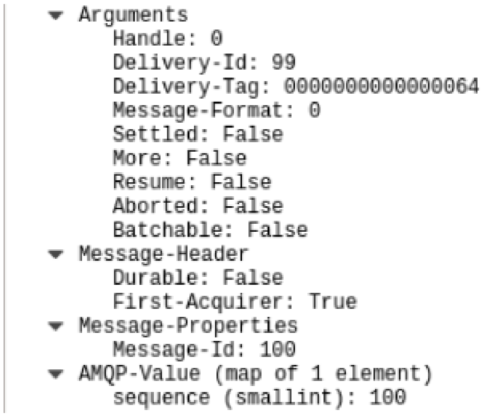


```
No.   Time         Source      Destination  Protocol Length Info
   35 7.458981472  127.0.1.1   127.0.0.1    AMQP      102 begin
   36 7.459487026  127.0.1.1   127.0.0.1    AMQP      216 attach flow
   38 7.469499421  127.0.0.1   127.0.1.1    AMQP     7258 transfer transfer transfer transfer transfer transfer transfer transfer transfer transfer transfer transfer transfer transfer transfer transfe…
   40 7.472320553  127.0.1.1   127.0.0.1    AMQP      149 transfer
   41 7.472724972  127.0.0.1   127.0.1.1    AMQP       95 disposition
   42 7.473138471  127.0.1.1   127.0.0.1    AMQP      150 transfer
   43 7.473391147  127.0.0.1   127.0.1.1    AMQP      137 flow disposition
   44 7.474833709  127.0.1.1   127.0.0.1    AMQP      150 transfer
   45 7.475106006  127.0.0.1   127.0.1.1    AMQP      137 flow disposition
   46 7.475358886  127.0.1.1   127.0.0.1    AMQP       89 disposition
   47 7.475486977  127.0.1.1   127.0.0.1    AMQP      150 transfer
   48 7.475734582  127.0.0.1   127.0.1.1    AMQP      137 flow disposition
   49 7.476717914  127.0.1.1   127.0.0.1    AMQP      266 disposition disposition disposition disposition disposition disposition disposition disposition
   50 7.477120893  127.0.1.1   127.0.0.1    AMQP       91 disposition
▸ Frame 40: 149 bytes on wire (1192 bits), 149 bytes captured (1192 bits) on interface 0
▸ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
▸ Internet Protocol Version 4, Src: 127.0.1.1, Dst: 127.0.0.1
▸ Transmission Control Protocol, Src Port: 5672, Dst Port: 51118, Seq: 545, Ack: 359, Len: 83
▾ Advanced Message Queueing Protocol
    Length: 83
    Doff: 2
    Type: AMQP (0)
    Channel: 0
    Performative: transfer (20)
  ▾ Arguments
      Handle: 0
      Delivery-Id: 0
      Delivery-Tag: 0000000000000001
      Message-Format: 0
      Settled: False
      More: False
      Resume: False
      Aborted: False
      Batchable: False
  ▾ Message-Header
      Durable: False
      First-Acquirer: True
  ▾ Message-Properties
      Message-Id: 1
  ▾ AMQP-Value (map of 1 element)
      sequence (smallint): 1

0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ........ ......E.
0010  00 87 3c 99 40 00 40 06  fe d5 7f 00 01 01 7f 00   ..<.@.@. ........
```



```
No.   Time         Source      Destination  Protocol Length Info
   33 7.458027406  127.0.1.1   127.0.0.1    AMQP       74 Protocol-Header 1-0-0
   34 7.458307718  127.0.1.1   127.0.0.1    AMQP      300 open
   35 7.458981472  127.0.1.1   127.0.0.1    AMQP      102 begin
   36 7.459487026  127.0.1.1   127.0.0.1    AMQP      216 attach flow
   38 7.469499421  127.0.0.1   127.0.1.1    AMQP     7258 transfer transfer transfer transfer transfer transfer transfer transfer transfer transfer transfer transfer transfer transfer transfer transfe…
   40 7.472320553  127.0.1.1   127.0.0.1    AMQP      149 transfer
   41 7.472724972  127.0.0.1   127.0.1.1    AMQP       95 disposition
   42 7.473138471  127.0.1.1   127.0.0.1    AMQP      150 transfer
   43 7.473391147  127.0.0.1   127.0.1.1    AMQP      137 flow disposition
   44 7.474833709  127.0.1.1   127.0.0.1    AMQP      150 transfer
   45 7.475106006  127.0.0.1   127.0.1.1    AMQP      137 flow disposition
   46 7.475358886  127.0.1.1   127.0.0.1    AMQP       89 disposition
   47 7.475486977  127.0.1.1   127.0.0.1    AMQP      150 transfer
   48 7.475734582  127.0.0.1   127.0.1.1    AMQP      137 flow disposition
▾ Frame 40: 149 bytes on wire (1192 bits), 149 bytes captured (1192 bits) on interface 0
  ▸ Interface id: 0 (lo)
    Encapsulation type: Ethernet (1)
    Arrival Time: Mar 29, 2023 16:41:17.805179632 CEST
    [Time shift for this packet: 0.000000000 seconds]
    Epoch Time: 1680100877.805179632 seconds
    [Time delta from previous captured frame: 0.002799992 seconds]
    [Time delta from previous displayed frame: 0.002821132 seconds]
    [Time since reference or first frame: 7.472320553 seconds]
    Frame Number: 40
    Frame Length: 149 bytes (1192 bits)
    Capture Length: 149 bytes (1192 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: eth:ethertype:ip:tcp:amqp]
    [Coloring Rule Name: TCP]
    [Coloring Rule String: tcp]
▸ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
▾ Internet Protocol Version 4, Src: 127.0.1.1, Dst: 127.0.0.1
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  ▸ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 135
    Identification: 0x3c99 (15513)
  ▸ Flags: 0x02 (Don't Fragment)
    Fragment offset: 0
    Time to live: 64
    Protocol: TCP (6)
    Header checksum: 0xfed5 [validation disabled]
    [Header checksum status: Unverified]
    Source: 127.0.1.1
    Destination: 127.0.0.1
    [Source GeoIP: Unknown]
    [Destination GeoIP: Unknown]

0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ........ ......E.
0010  00 87 3c 99 40 00 40 06  fe d5 7f 00 01 01 7f 00   ..<.@.@. ........
```

Also, there are 100 messages present:



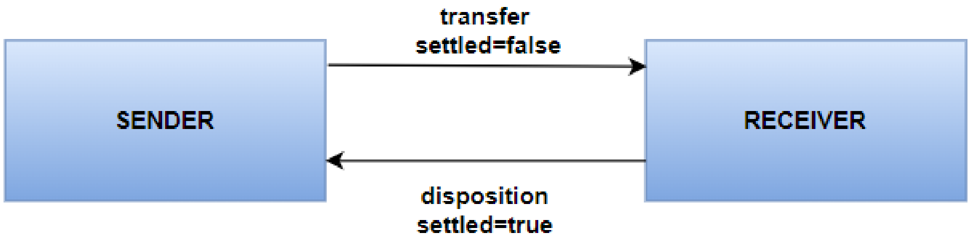
```
▾ Arguments
     Handle: 0
     Delivery-Id: 99
     Delivery-Tag: 0000000000000064
     Message-Format: 0
     Settled: False
     More: False
     Resume: False
     Aborted: False
     Batchable: False
▾ Message-Header
     Durable: False
     First-Acquirer: True
▾ Message-Properties
     Message-Id: 100
▾ AMQP-Value (map of 1 element)
     sequence (smallint): 100
```

In the sender.py code, the below line is used to define an option for the script that takes an integer value and is used to set the number of messages to be sent by the sender. By default, this value is 100.

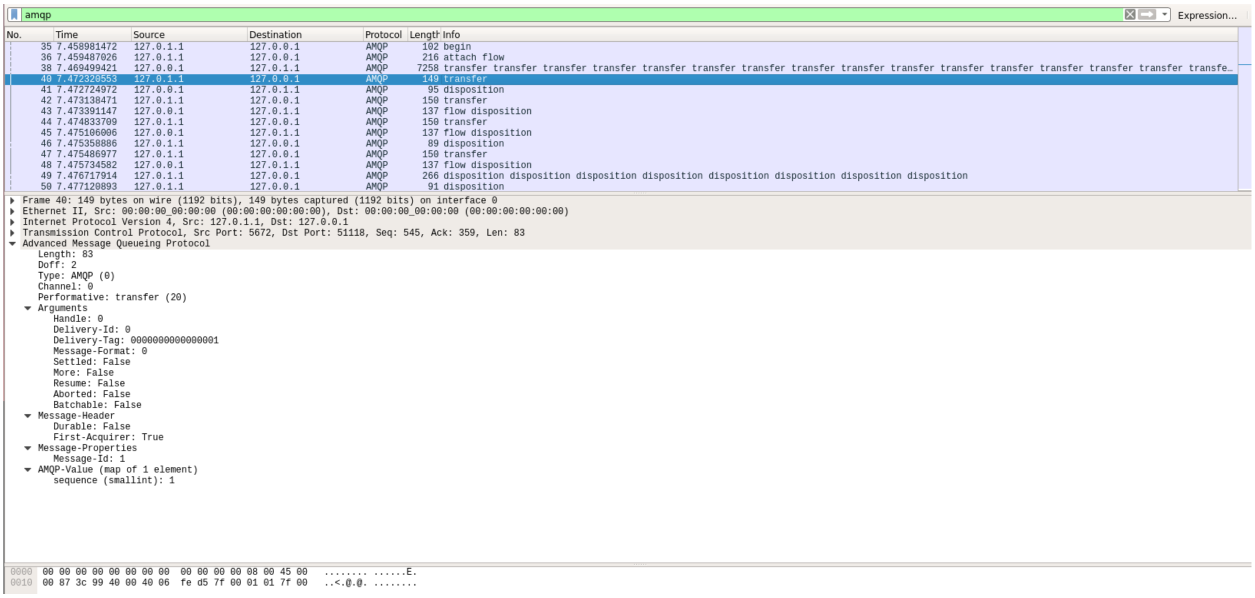
```
parser.add_option ("-m","--messages", type="int", default=100,
               help="number of messages to send (default %default)")
```

12. What is the type of QoS? (Hint: have a look to transfer and disposition messages).

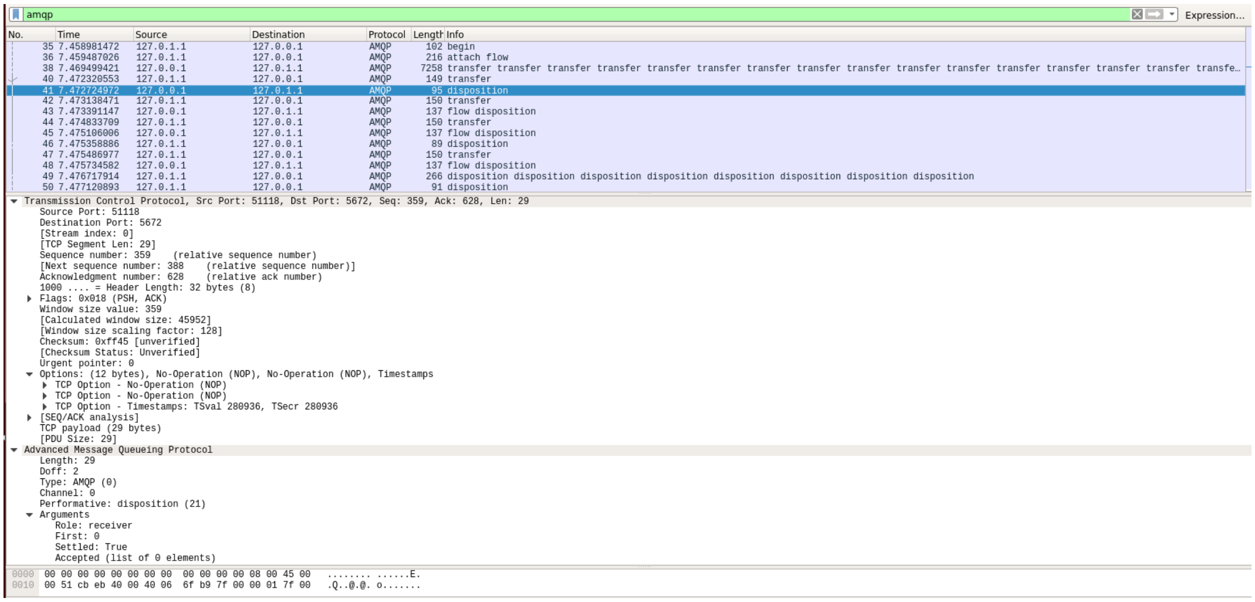
The QoS is 1 (at-least once). Sender sends delivery with settled = false and only settle when settled = true disposition received from the receiver.



Below are the Wireshark captures where we can see the **"Settled"** values for the transfer and disposition packets:



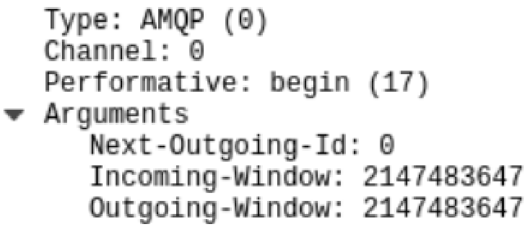
```
No.  Time         Source      Destination  Protocol Length Info
  35 7.458981472  127.0.1.1   127.0.0.1    AMQP      102 begin
  36 7.459487026  127.0.1.1   127.0.0.1    AMQP      216 attach flow
  38 7.469499421  127.0.0.1   127.0.1.1    AMQP     7258 transfer transfer transfer transfer transfer transfer transfer transfer transfer transfer transfer transfer transfer transfer transfer transfe…
  40 7.472320553  127.0.1.1   127.0.0.1    AMQP      149 transfer
  41 7.472724972  127.0.0.1   127.0.1.1    AMQP       95 disposition
  42 7.473138471  127.0.1.1   127.0.0.1    AMQP      150 transfer
  43 7.473391147  127.0.0.1   127.0.1.1    AMQP      137 flow disposition
  44 7.474833709  127.0.1.1   127.0.0.1    AMQP      150 transfer
  45 7.475106006  127.0.0.1   127.0.1.1    AMQP      137 flow disposition
  46 7.475358886  127.0.1.1   127.0.0.1    AMQP       89 disposition
  47 7.475486977  127.0.1.1   127.0.0.1    AMQP      150 transfer
  48 7.475734582  127.0.0.1   127.0.1.1    AMQP      137 flow disposition
  49 7.476717914  127.0.1.1   127.0.0.1    AMQP      266 disposition disposition disposition disposition disposition disposition disposition disposition
  50 7.477120893  127.0.1.1   127.0.0.1    AMQP       91 disposition
▶ Frame 40: 149 bytes on wire (1192 bits), 149 bytes captured (1192 bits) on interface 0
▶ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
▶ Internet Protocol Version 4, Src: 127.0.1.1, Dst: 127.0.0.1
▶ Transmission Control Protocol, Src Port: 5672, Dst Port: 51118, Seq: 545, Ack: 359, Len: 83
▼ Advanced Message Queueing Protocol
    Length: 83
    Doff: 2
    Type: AMQP (0)
    Channel: 0
    Performative: transfer (20)
  ▼ Arguments
      Handle: 0
      Delivery-Id: 0
      Delivery-Tag: 0000000000000001
      Message-Format: 0
      Settled: False
      More: False
      Resume: False
      Aborted: False
      Batchable: False
  ▼ Message-Header
      Durable: False
      First-Acquirer: True
  ▼ Message-Properties
      Message-Id: 1
  ▼ AMQP-Value (map of 1 element)
      sequence (smallint): 1

0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ........ ......E.
0010  00 87 3c 99 40 00 40 06  fe d5 7f 00 01 01 7f 00   ..<.@.@. ........
```



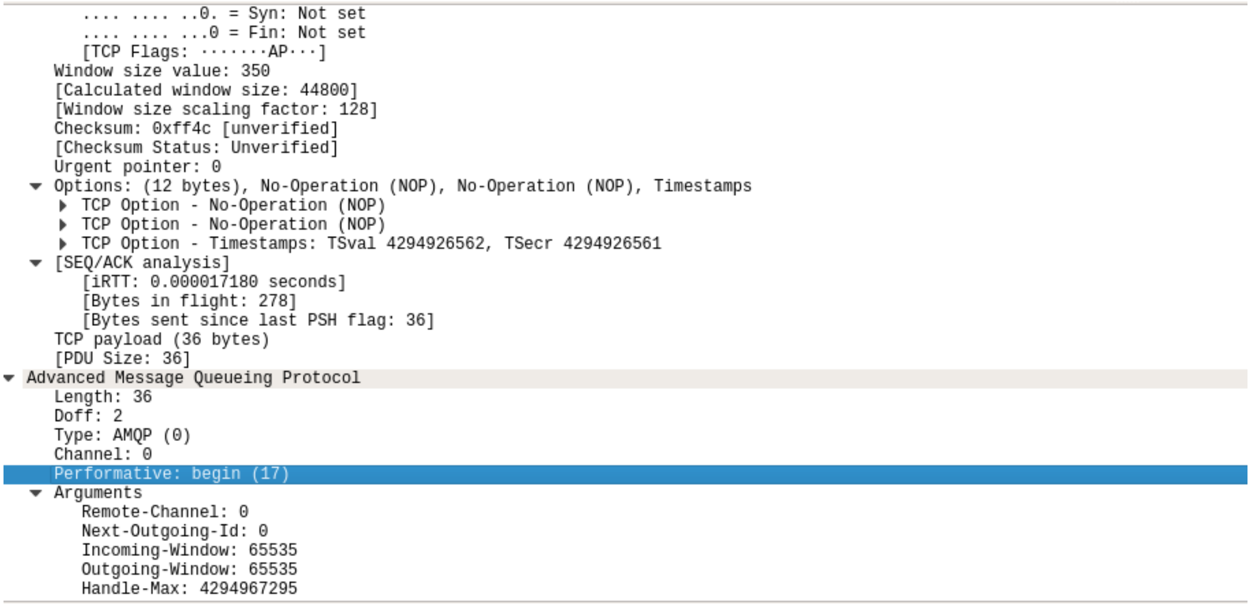
```
No.  Time         Source      Destination  Protocol Length Info
  35 7.458981472  127.0.1.1   127.0.0.1    AMQP      102 begin
  36 7.459487026  127.0.1.1   127.0.0.1    AMQP      216 attach flow
  38 7.469499421  127.0.0.1   127.0.1.1    AMQP     7258 transfer transfer transfer transfer transfer transfer transfer transfer transfer transfer transfer transfer transfer transfer transfer transfe…
  40 7.472320553  127.0.1.1   127.0.0.1    AMQP      149 transfer
  41 7.472724972  127.0.0.1   127.0.1.1    AMQP       95 disposition
  42 7.473138471  127.0.1.1   127.0.0.1    AMQP      150 transfer
  43 7.473391147  127.0.0.1   127.0.1.1    AMQP      137 flow disposition
  44 7.474833709  127.0.1.1   127.0.0.1    AMQP      150 transfer
  45 7.475106006  127.0.0.1   127.0.1.1    AMQP      137 flow disposition
  46 7.475358886  127.0.1.1   127.0.0.1    AMQP       89 disposition
  47 7.475486977  127.0.1.1   127.0.0.1    AMQP      150 transfer
  48 7.475734582  127.0.0.1   127.0.1.1    AMQP      137 flow disposition
  49 7.476717914  127.0.1.1   127.0.0.1    AMQP      266 disposition disposition disposition disposition disposition disposition disposition disposition
  50 7.477120893  127.0.1.1   127.0.0.1    AMQP       91 disposition
▼ Transmission Control Protocol, Src Port: 51118, Dst Port: 5672, Seq: 359, Ack: 628, Len: 29
    Source Port: 51118
    Destination Port: 5672
    [Stream index: 0]
    [TCP Segment Len: 29]
    Sequence number: 359    (relative sequence number)
    [Next sequence number: 388    (relative sequence number)]
    Acknowledgment number: 628    (relative ack number)
    1000 .... = Header Length: 32 bytes (8)
  ▶ Flags: 0x018 (PSH, ACK)
    Window size value: 359
    [Calculated window size: 45952]
    [Window size scaling factor: 128]
    Checksum: 0xff45 [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
  ▼ Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
    ▶ TCP Option - No-Operation (NOP)
    ▶ TCP Option - No-Operation (NOP)
    ▶ TCP Option - Timestamps: TSval 280936, TSecr 280936
  ▶ [SEQ/ACK analysis]
    TCP payload (29 bytes)
    [PDU Size: 29]
▼ Advanced Message Queueing Protocol
    Length: 29
    Doff: 2
    Type: AMQP (0)
    Channel: 0
    Performative: disposition (21)
  ▼ Arguments
      Role: receiver
      First: 0
      Settled: True
      Accepted (list of 0 elements)
0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ........ ......E.
0010  00 51 cb eb 40 00 40 06  6f b9 7f 00 00 01 7f 00   .Q..@.@. o.......
```

13. Check the window size evolution. What do you remark? Why?

In the packet “**open begin attach**”, the window size is 2147483647. In the **“begin”** packet, the window size is 65535. This tells the sender that the exchange is capable of receiving this in/out window size and the sender has to reduce its window size.

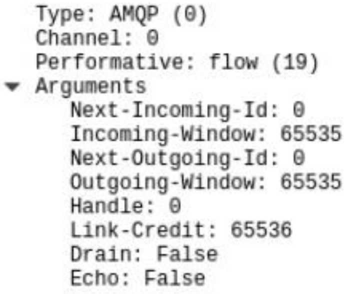


```
    Type: AMQP (0)
    Channel: 0
    Performative: begin (17)
  ▼ Arguments
      Next-Outgoing-Id: 0
      Incoming-Window: 2147483647
      Outgoing-Window: 2147483647
```



```
        .... .... ..0. = Syn: Not set
        .... .... ...0 = Fin: Not set
        [TCP Flags: ·······AP···]
     Window size value: 350
     [Calculated window size: 44800]
     [Window size scaling factor: 128]
     Checksum: 0xff4c [unverified]
     [Checksum Status: Unverified]
     Urgent pointer: 0
   ▼ Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
      ▶ TCP Option - No-Operation (NOP)
      ▶ TCP Option - No-Operation (NOP)
      ▶ TCP Option - Timestamps: TSval 4294926562, TSecr 4294926561
   ▼ [SEQ/ACK analysis]
        [iRTT: 0.000017180 seconds]
        [Bytes in flight: 278]
        [Bytes sent since last PSH flag: 36]
     TCP payload (36 bytes)
     [PDU Size: 36]
▼ Advanced Message Queueing Protocol
     Length: 36
     Doff: 2
     Type: AMQP (0)
     Channel: 0
     Performative: begin (17)
   ▼ Arguments
        Remote-Channel: 0
        Next-Outgoing-Id: 0
        Incoming-Window: 65535
        Outgoing-Window: 65535
        Handle-Max: 4294967295
```

In the next message, the container adjusts the value of the in/out window to match the previous value.

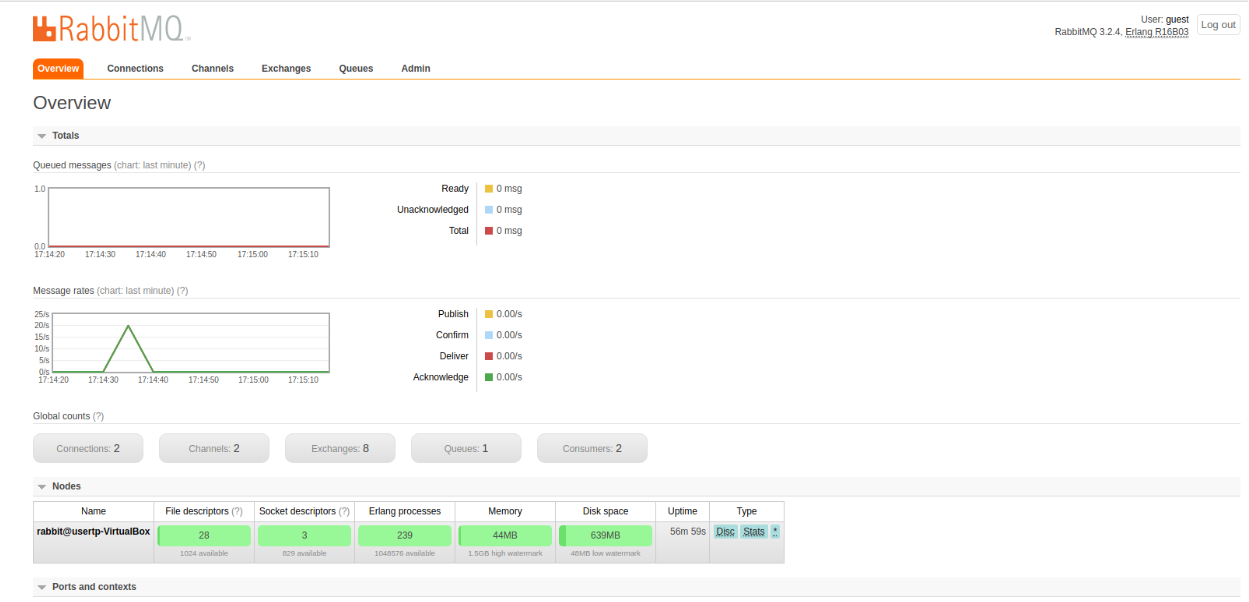


```
   Type: AMQP (0)
   Channel: 0
   Performative: flow (19)
▼ Arguments
      Next-Incoming-Id: 0
      Incoming-Window: 65535
      Next-Outgoing-Id: 0
      Outgoing-Window: 65535
      Handle: 0
      Link-Credit: 65536
      Drain: False
      Echo: False
```

14. RabbitMQ comes with a web-based management interface. You may enter the following URL:

http://localhost: 15672.

Please refer to the documentation available on the website of RabbitMQ to see the features available with the management interface. The account to use is login: guest, passwd: guest. Add another receiver, and start it.



Overview

Totals

Queued messages (chart: last minute) (?)

Ready	0 msg
Unacknowledged	0 msg
Total	0 msg

Message rates (chart: last minute) (?)

Publish	0.00/s
Confirm	0.00/s
Deliver	0.00/s
Acknowledge	0.00/s

Global counts (?)

Connections: 2 Channels: 2 Exchanges: 8 Queues: 1 Consumers: 2

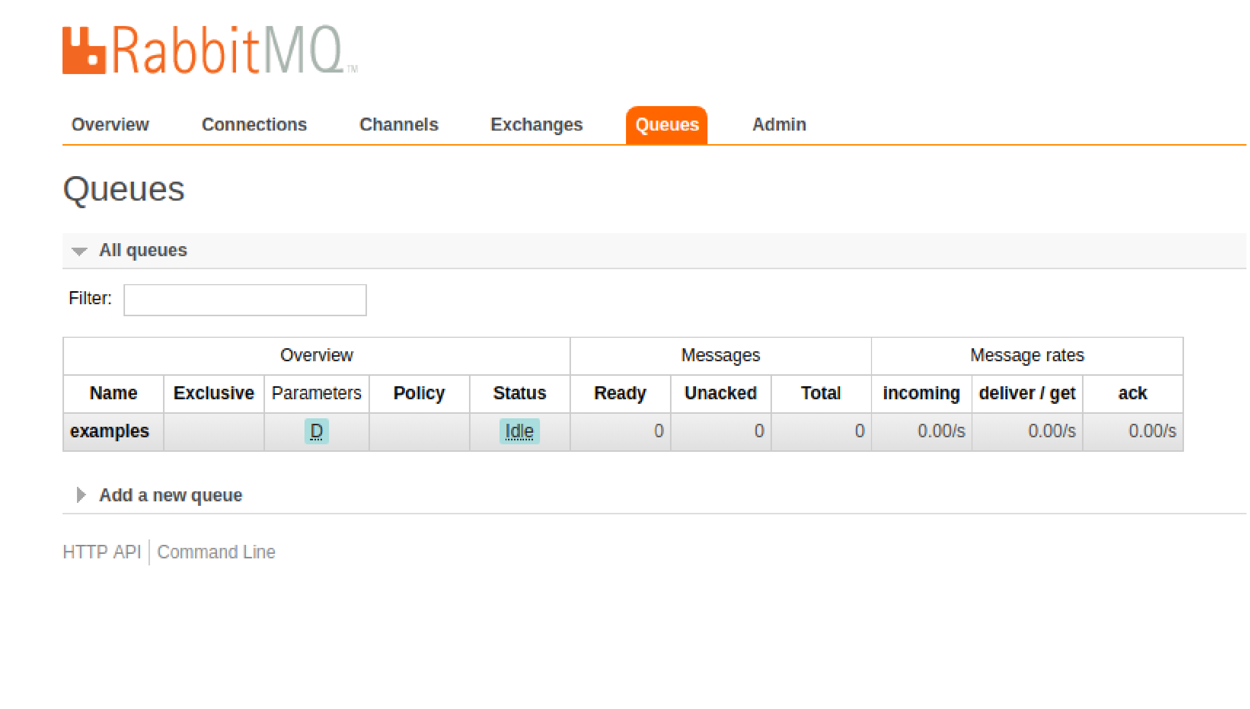
Nodes

Name	File descriptors (?)	Socket descriptors (?)	Erlang processes	Memory	Disk space	Uptime	Type
rabbit@usertp-VirtualBox	28 1024 available	3 829 available	239 1048576 available	44MB 1.5GB high watermark	639MB 48MB low watermark	56m 59s	Disc Stats *

Ports and contexts

15. Check in RabbitMQ the status of the queues? What is the name of the queue used by the receiver and sender? Which type of binding is used?

The name of the queue is “**examples**” and the status is “**Idle”.** The binding type used here is the default exchange binding. The message is routed to the queue based on the queue name. If the queue name is not specified, then the message is routed to a queue with the same name as the routing key. This binding type is often used when the receiver and the sender are both using the same queue.



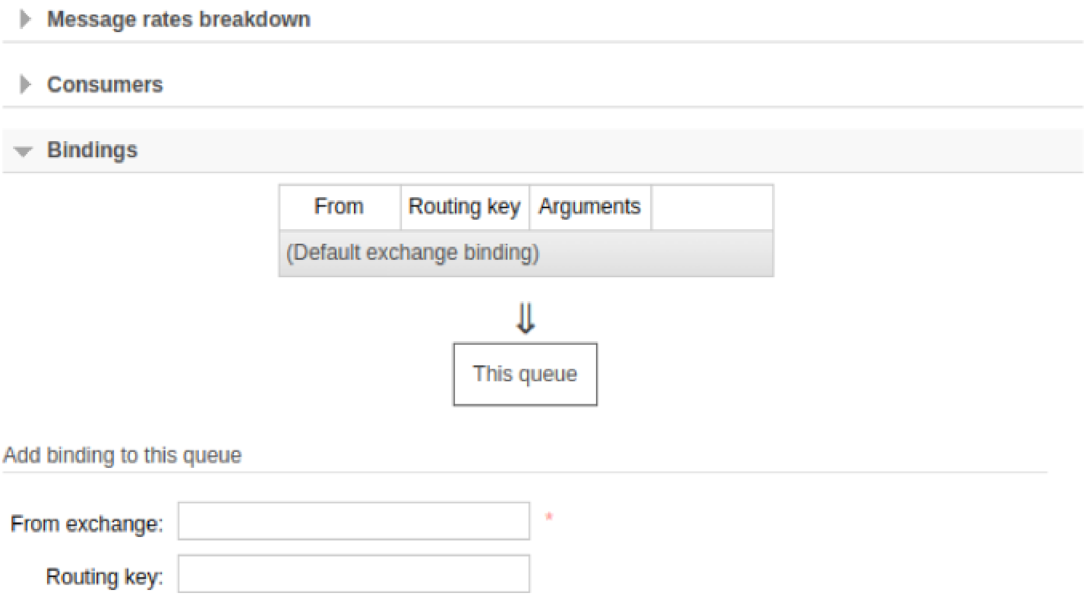
Queues

All queues

Filter:

Overview					Messages			Message rates		
Name	Exclusive	Parameters	Policy	Status	Ready	Unacked	Total	incoming	deliver / get	ack
examples		D		Idle	0	0	0	0.00/s	0.00/s	0.00/s

Add a new queue



From	Routing key	Arguments	
(Default exchange binding)			

⇓

This queue

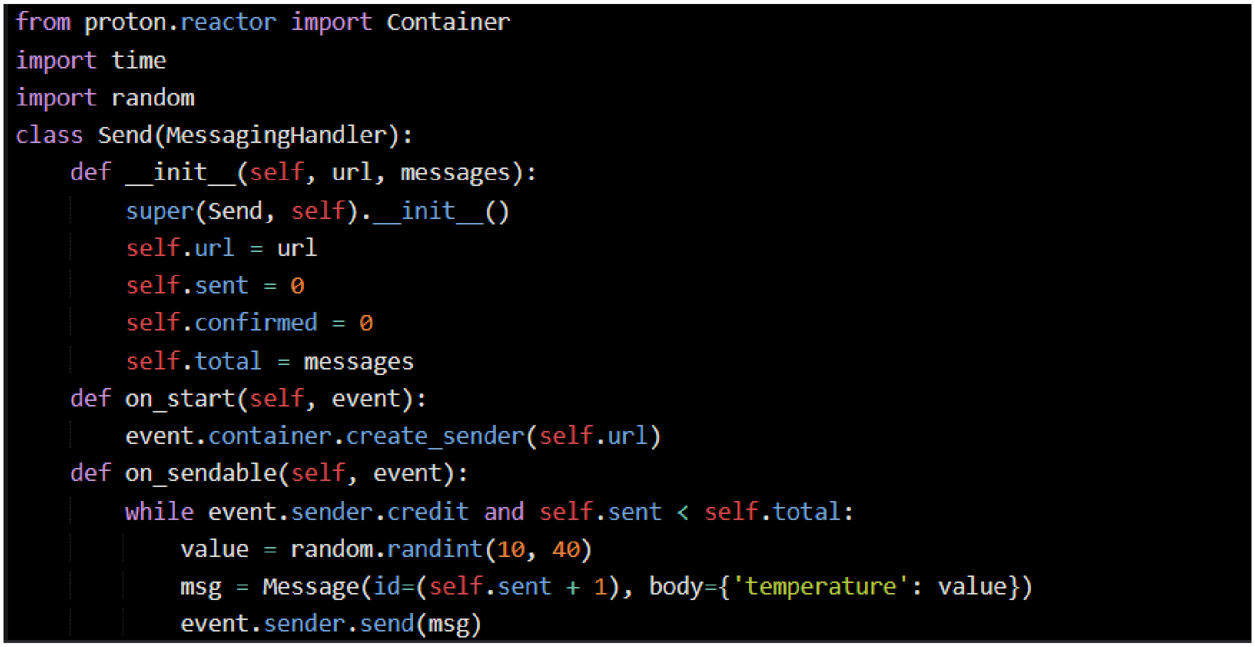
Add binding to this queue

From exchange: *

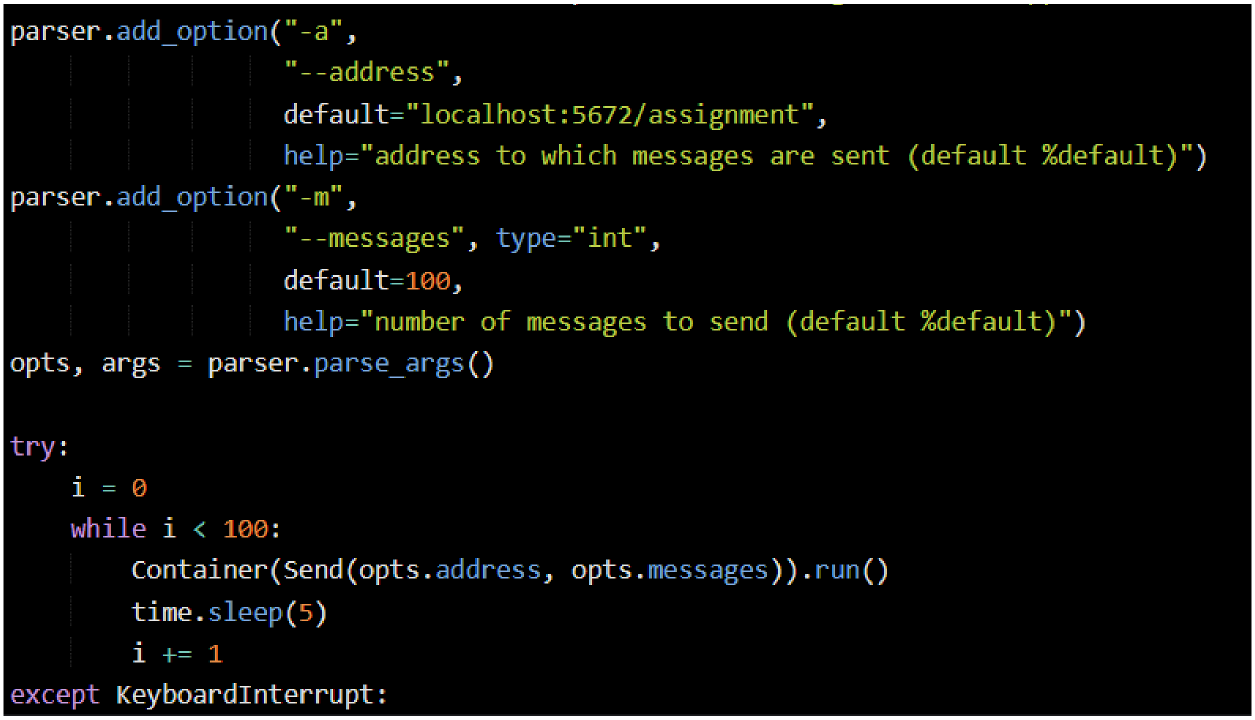
Routing key:

B) PART-2

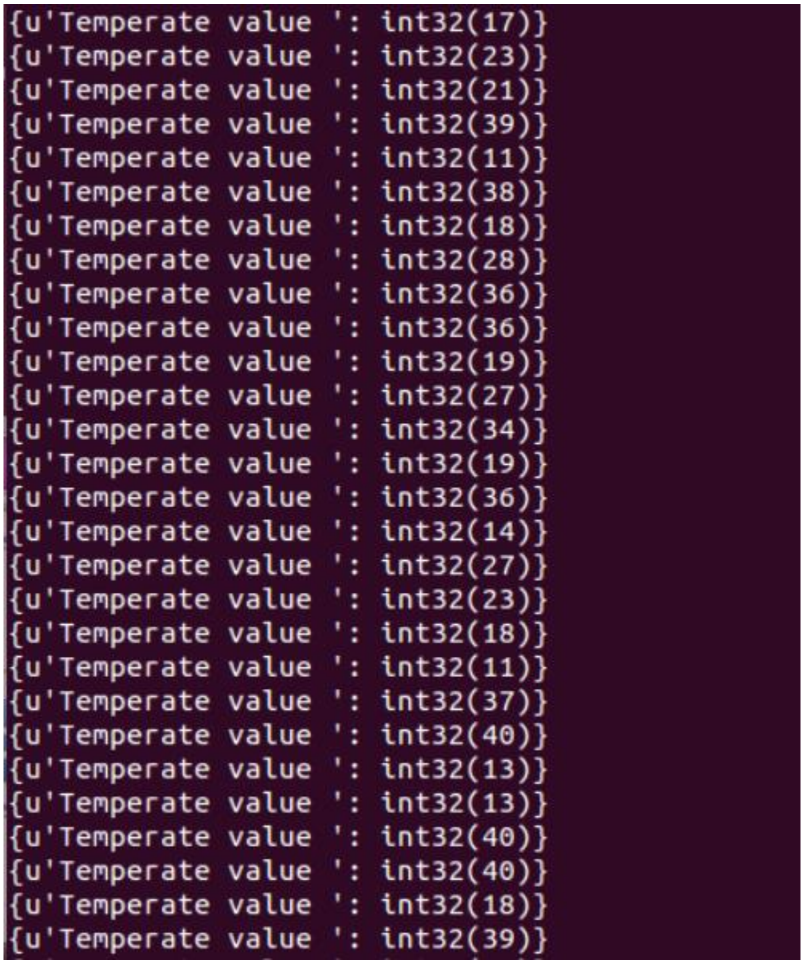
1. Modifying the sender to publish not a batch of messages, but generate a random value periodically using temperature as a topic.



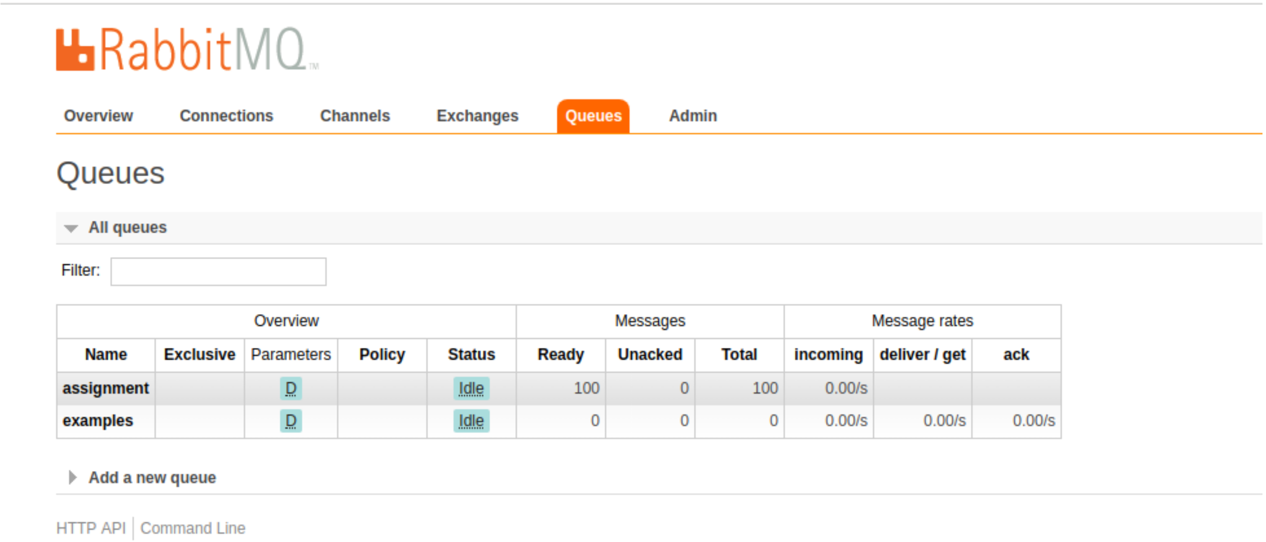
```
from proton.reactor import Container
import time
import random
class Send(MessagingHandler):
    def __init__(self, url, messages):
        super(Send, self).__init__()
        self.url = url
        self.sent = 0
        self.confirmed = 0
        self.total = messages
    def on_start(self, event):
        event.container.create_sender(self.url)
    def on_sendable(self, event):
        while event.sender.credit and self.sent < self.total:
            value = random.randint(10, 40)
            msg = Message(id=(self.sent + 1), body={'temperature': value})
            event.sender.send(msg)
```



```
parser.add_option("-a",
                  "--address",
                  default="localhost:5672/assignment",
                  help="address to which messages are sent (default %default)")
parser.add_option("-m",
                  "--messages", type="int",
                  default=100,
                  help="number of messages to send (default %default)")
opts, args = parser.parse_args()

try:
    i = 0
    while i < 100:
        Container(Send(opts.address, opts.messages)).run()
        time.sleep(5)
        i += 1
except KeyboardInterrupt:
```



```
{u'Temperate value ': int32(17)}
{u'Temperate value ': int32(23)}
{u'Temperate value ': int32(21)}
{u'Temperate value ': int32(39)}
{u'Temperate value ': int32(11)}
{u'Temperate value ': int32(38)}
{u'Temperate value ': int32(18)}
{u'Temperate value ': int32(28)}
{u'Temperate value ': int32(36)}
{u'Temperate value ': int32(36)}
{u'Temperate value ': int32(19)}
{u'Temperate value ': int32(27)}
{u'Temperate value ': int32(34)}
{u'Temperate value ': int32(19)}
{u'Temperate value ': int32(36)}
{u'Temperate value ': int32(14)}
{u'Temperate value ': int32(27)}
{u'Temperate value ': int32(23)}
{u'Temperate value ': int32(18)}
{u'Temperate value ': int32(11)}
{u'Temperate value ': int32(37)}
{u'Temperate value ': int32(40)}
{u'Temperate value ': int32(13)}
{u'Temperate value ': int32(13)}
{u'Temperate value ': int32(40)}
{u'Temperate value ': int32(40)}
{u'Temperate value ': int32(18)}
{u'Temperate value ': int32(39)}
```



Queues

All queues

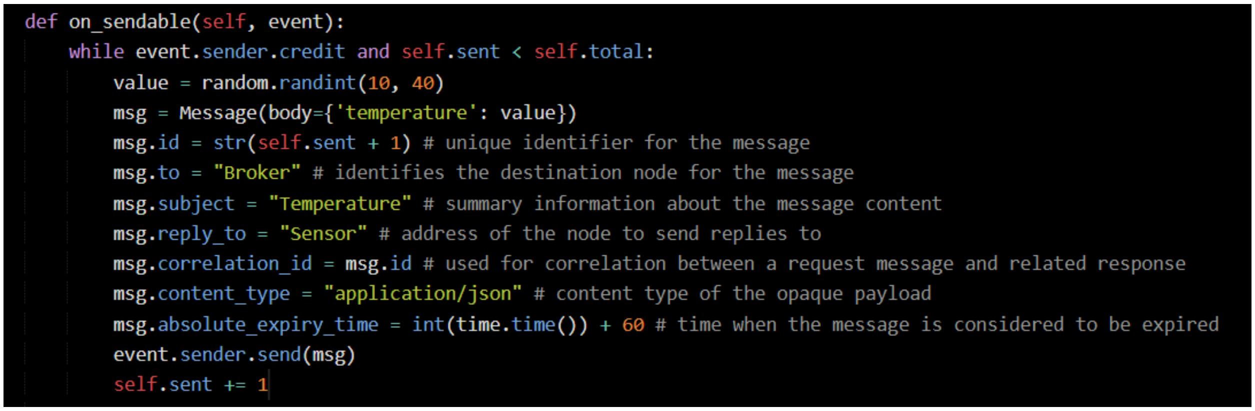
Filter:

Overview					Messages			Message rates		
Name	Exclusive	Parameters	Policy	Status	Ready	Unacked	Total	incoming	deliver / get	ack
assignment		D		Idle	100	0	100	0.00/s		
examples		D		Idle	0	0	0	0.00/s	0.00/s	0.00/s

Add a new queue

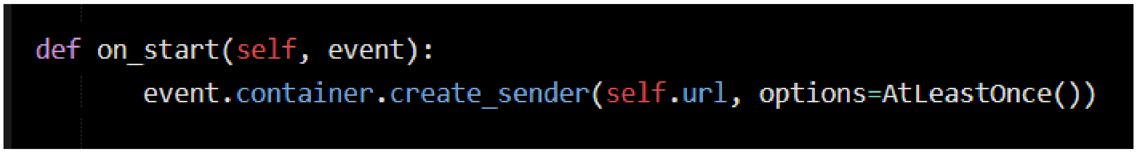
HTTP API | Command Line

2. Build the messages using the fields seen in the lecture on AMQP.



```
def on_sendable(self, event):
    while event.sender.credit and self.sent < self.total:
        value = random.randint(10, 40)
        msg = Message(body={'temperature': value})
        msg.id = str(self.sent + 1) # unique identifier for the message
        msg.to = "Broker" # identifies the destination node for the message
        msg.subject = "Temperature" # summary information about the message content
        msg.reply_to = "Sensor" # address of the node to send replies to
        msg.correlation_id = msg.id # used for correlation between a request message and related response
        msg.content_type = "application/json" # content type of the opaque payload
        msg.absolute_expiry_time = int(time.time()) + 60 # time when the message is considered to be expired
        event.sender.send(msg)
        self.sent += 1
```

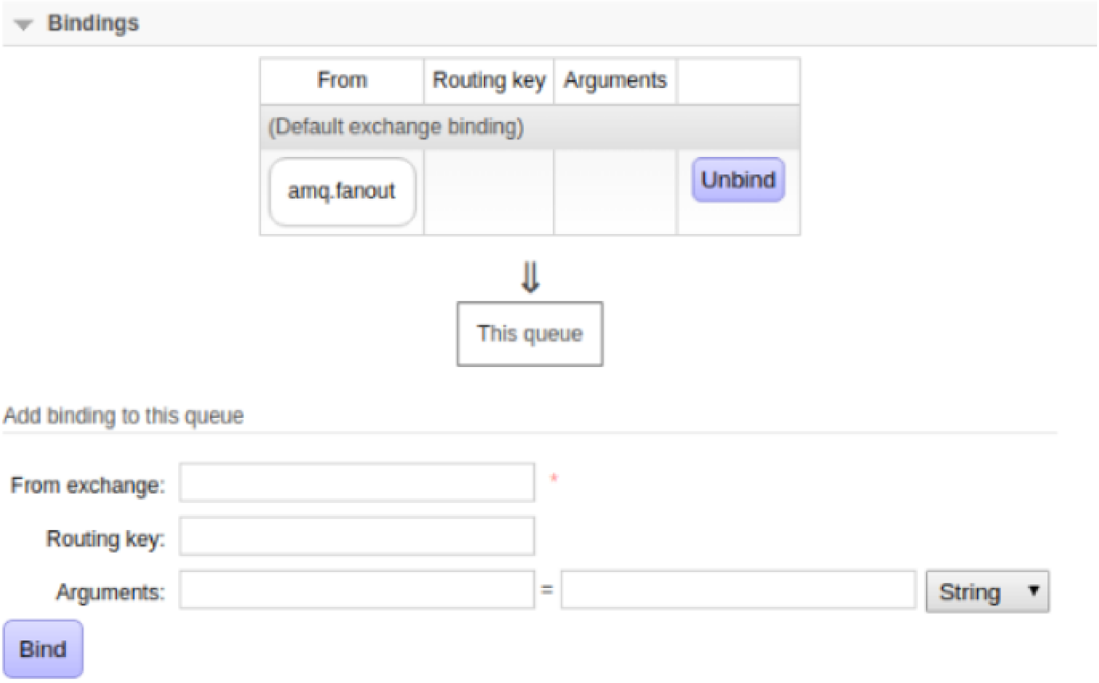
3. Modify the QoS to at least once.



```
def on_start(self, event):
    event.container.create_sender(self.url, options=AtLeastOnce())
```

4. Try to use different binding at the RabbitMQ.

The name of the exchange = amq.fanout binding



Bindings

From	Routing key	Arguments	
(Default exchange binding)			
amq.fanout			Unbind

⇓

This queue

Add binding to this queue

From exchange:		*		
Routing key:				
Arguments:		=		String ▾

Bind