

MYE017 Distributed Systems

Kostas Magoutis

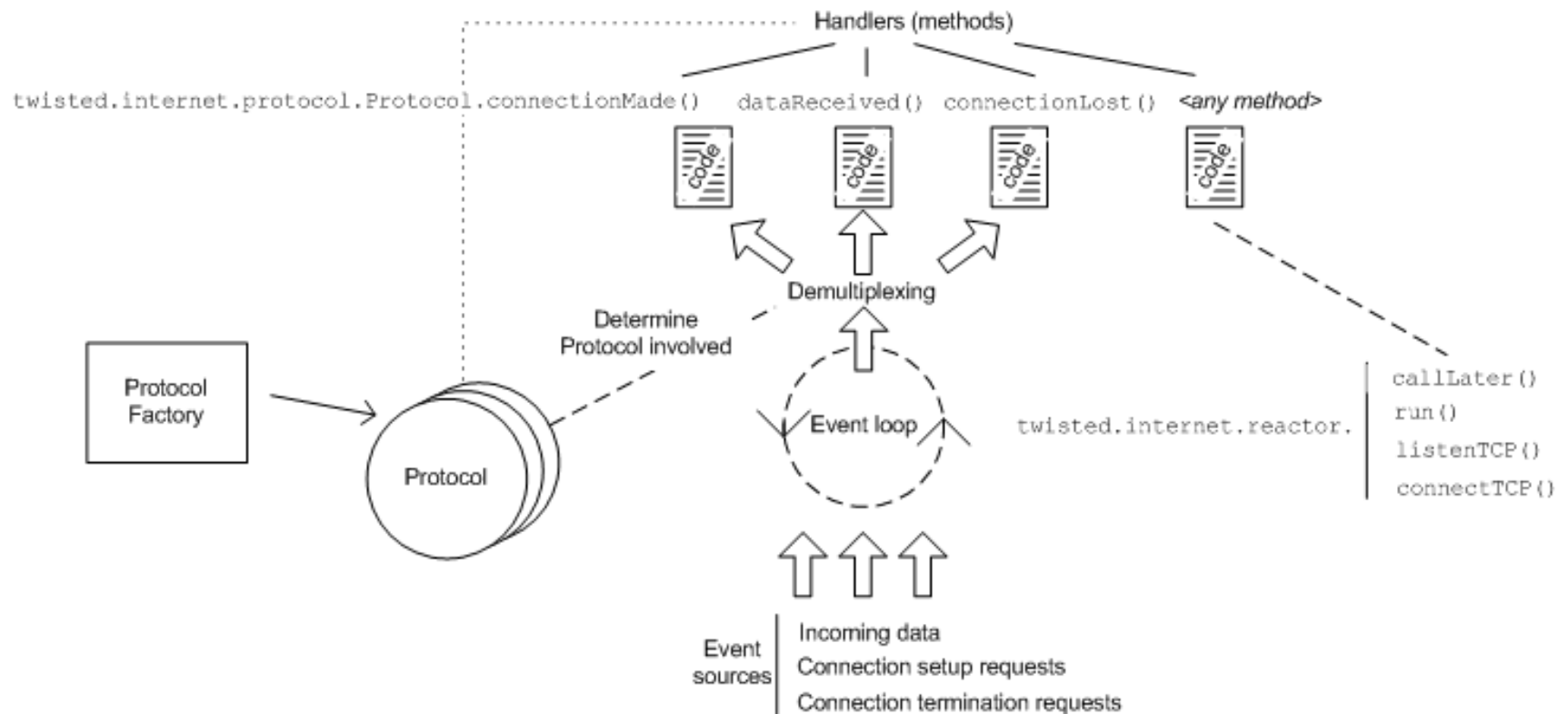
magoutis@cse.uoi.gr

<http://www.cse.uoi.gr/~magoutis>

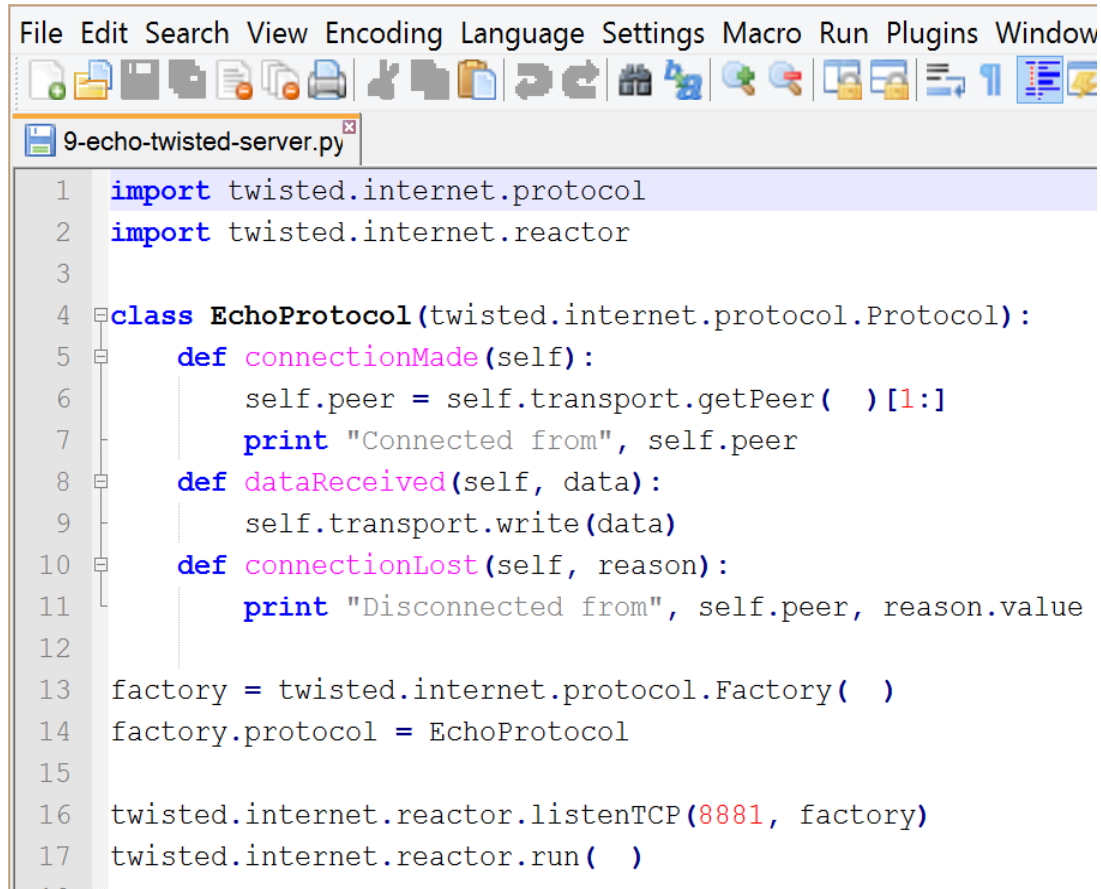
Twisted

- Event-driven communication framework
- Python module
- Rapid prototyping of distributed applications

Twisted event loop



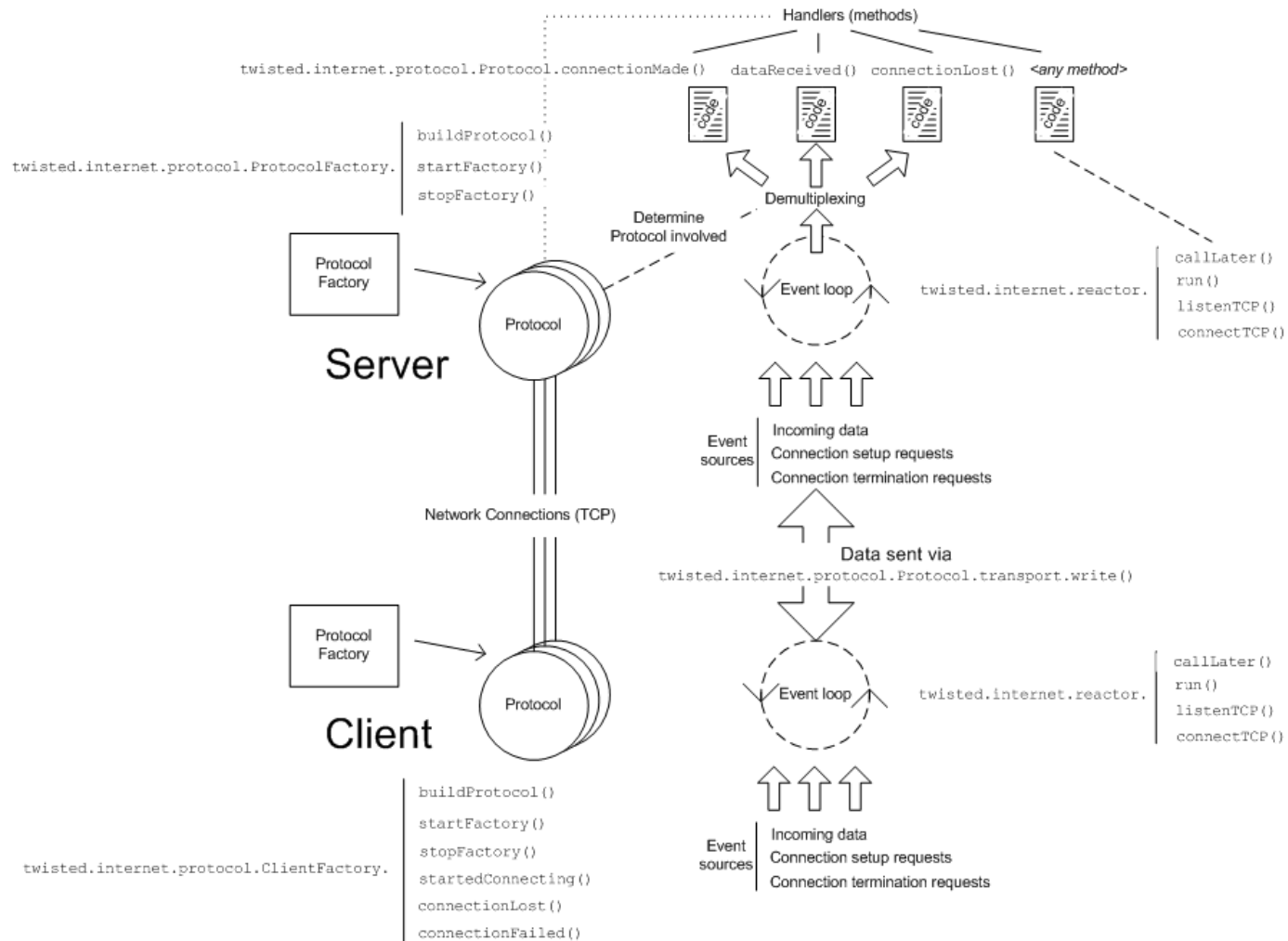
Echo server



The image shows a screenshot of an IDE window titled '9-echo-twisted-server.py'. The window contains Python code for a Twisted echo server. The code is as follows:

```
1 import twisted.internet.protocol
2 import twisted.internet.reactor
3
4 class EchoProtocol(twisted.internet.protocol.Protocol):
5     def connectionMade(self):
6         self.peer = self.transport.getPeer()[1:]
7         print "Connected from", self.peer
8     def dataReceived(self, data):
9         self.transport.write(data)
10    def connectionLost(self, reason):
11        print "Disconnected from", self.peer, reason.value
12
13    factory = twisted.internet.protocol.Factory()
14    factory.protocol = EchoProtocol
15
16    twisted.internet.reactor.listenTCP(8881, factory)
17    twisted.internet.reactor.run()
```

Twisted event loop (2)



Peer protocol

```
47 class Peer(Protocol):
48
49     acks = 0
50     connected = False
51
52     def __init__(self, factory, peer_type):
53         self.pt = peer_type
54         self.factory = factory
55
56     def connectionMade(self):
57         if self.pt == 'client':
58             self.connected = True
59             reactor.callLater(5, self.sendUpdate)
60         else:
61             print "Connected from", self.transport.client
62             try:
63                 self.transport.write('<connection up>')
64             except Exception, e:
65                 print e.args[0]
66             self.ts = time.time()
67
68     def sendUpdate(self):
69         print "Sending update"
70         try:
71             self.transport.write('<update>')
72         except Exception, ex1:
73             print "Exception trying to send: ", ex1.args[0]
74         if self.connected == True:
75             reactor.callLater(5, self.sendUpdate)
76
```

```
76
77 def sendAck(self):
78     print "sendAck"
79     self.ts = time.time()
80     try:
81         self.transport.write('<Ack>')
82     except Exception, e:
83         print e.args[0]
84
85     def dataReceived(self, data):
86         if self.pt == 'client':
87             print 'Client received ' + data
88             self.acks += 1
89         else:
90             print 'Server received ' + data
91             self.sendAck()
92
93     def connectionLost(self, reason):
94         print "Disconnected"
95         if self.pt == 'client':
96             self.connected = False
97             self.done()
98
99     def done(self):
100         self.factory.finished(self.acks)
101
```

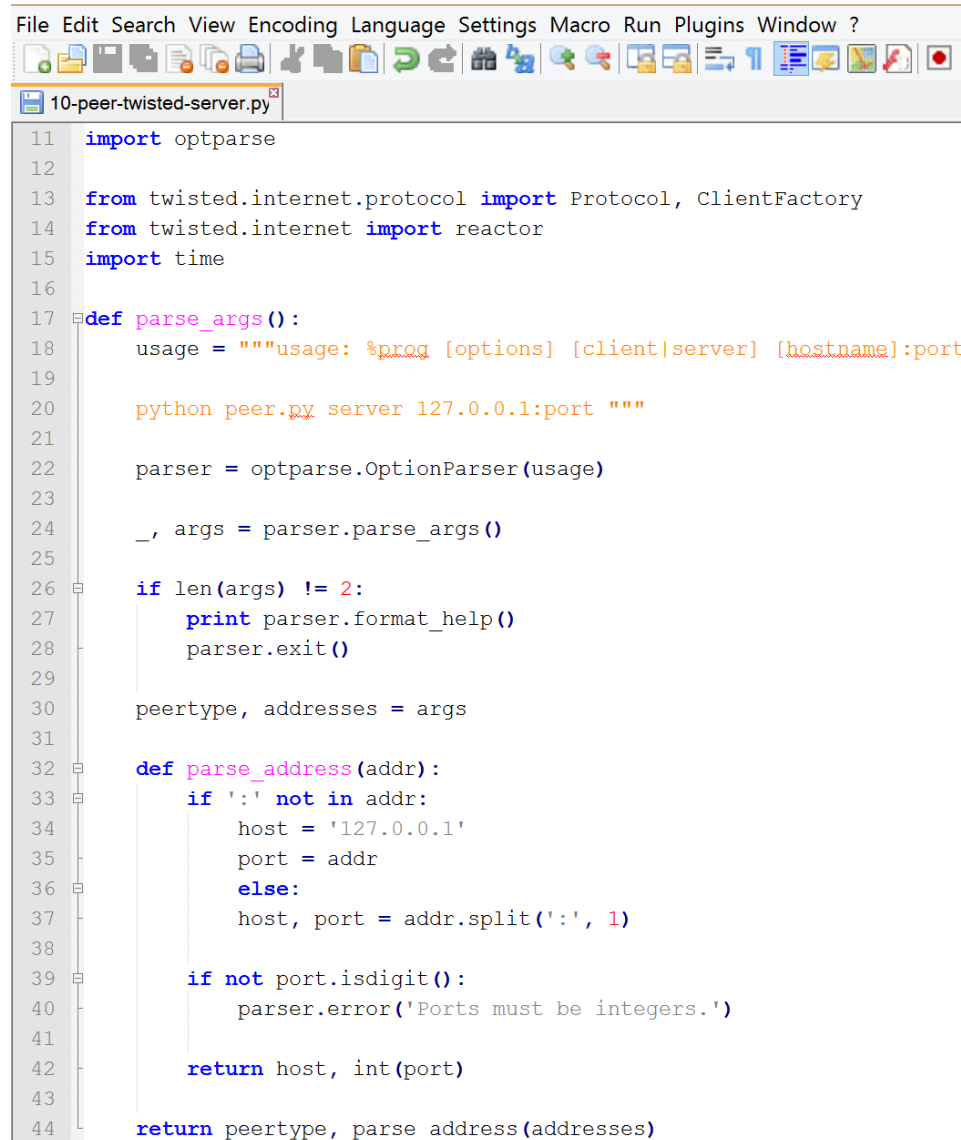
Main program

```
142 if __name__ == '__main__':
143     peer_type, address = parse_args()
144
145
146     if peer_type == 'server':
147         factory = PeerFactory('server', 'log')
148         reactor.listenTCP(8888, factory)
149         print "Starting server @" + address[0] + " port " + str(address[1])
150     else:
151         factory = PeerFactory('client', '')
152         host, port = address
153         print "Connecting to host " + host + " port " + str(port)
154         reactor.connectTCP(host, port, factory)
155
156     reactor.run()
```

Peer factory

```
103 class PeerFactory(ClientFactory):
104
105     def __init__(self, peertype, fname):
106         print '@__init__'
107         self.pt = peertype
108         self.acks = 0
109         self.fname = fname
110         self.records = []
111
112     def finished(self, arg):
113         self.acks = arg
114         self.report()
115
116     def report(self):
117         print 'Received %d acks' % self.acks
118
119     def clientConnectionFailed(self, connector, reason):
120         print 'Failed to connect to:', connector.getDestination()
121         self.finished(0)
122
123     def clientConnectionLost(self, connector, reason):
124         print 'Lost connection. Reason:', reason
125
126     def startFactory(self):
127         print "@startFactory"
128         if self.pt == 'server':
129             self.fp = open(self.fname, 'w+')
130
131     def stopFactory(self):
132         print "@stopFactory"
133         if self.pt == 'server':
134             self.fp.close()
135
136     def buildProtocol(self, addr):
137         print "@buildProtocol"
138         protocol = Peer(self, self.pt)
139         return protocol
140
```

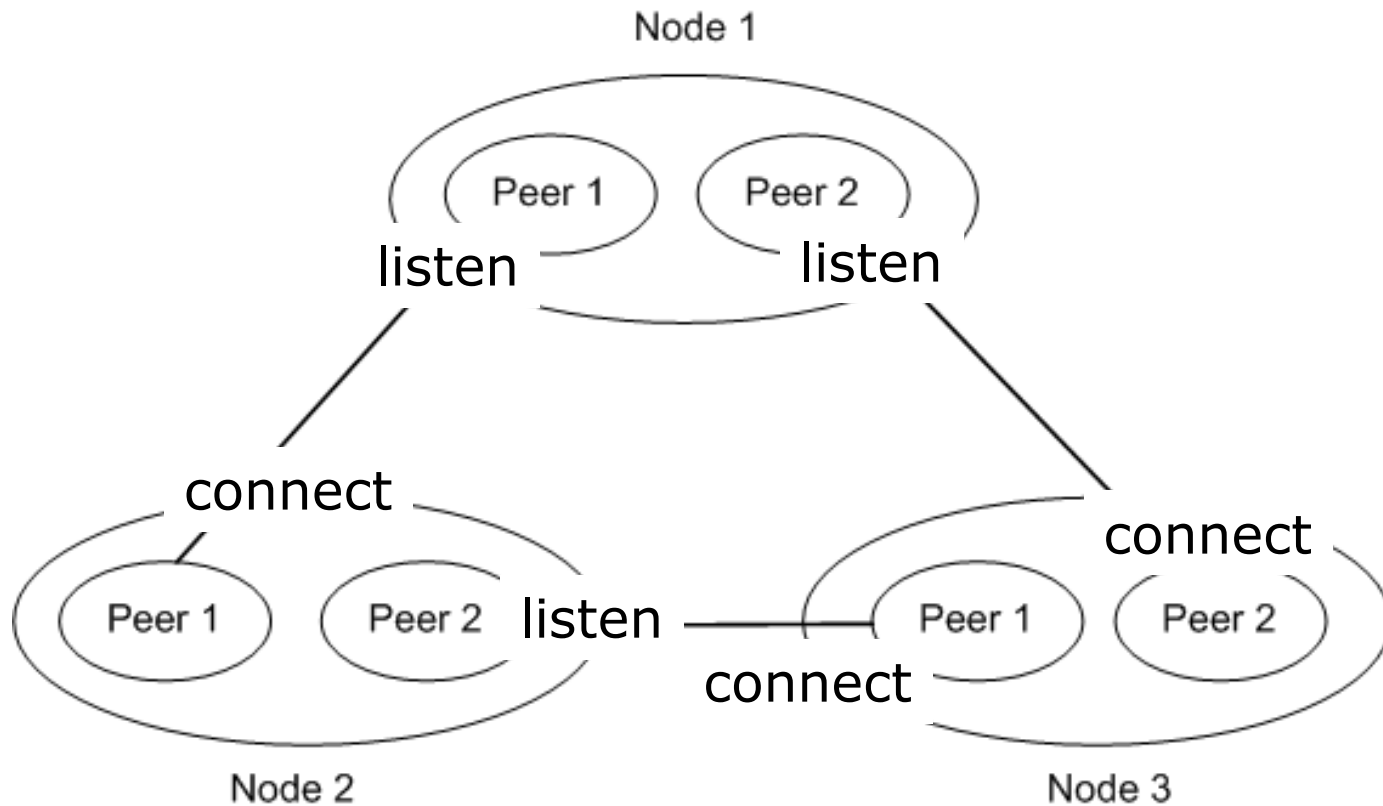

Program options



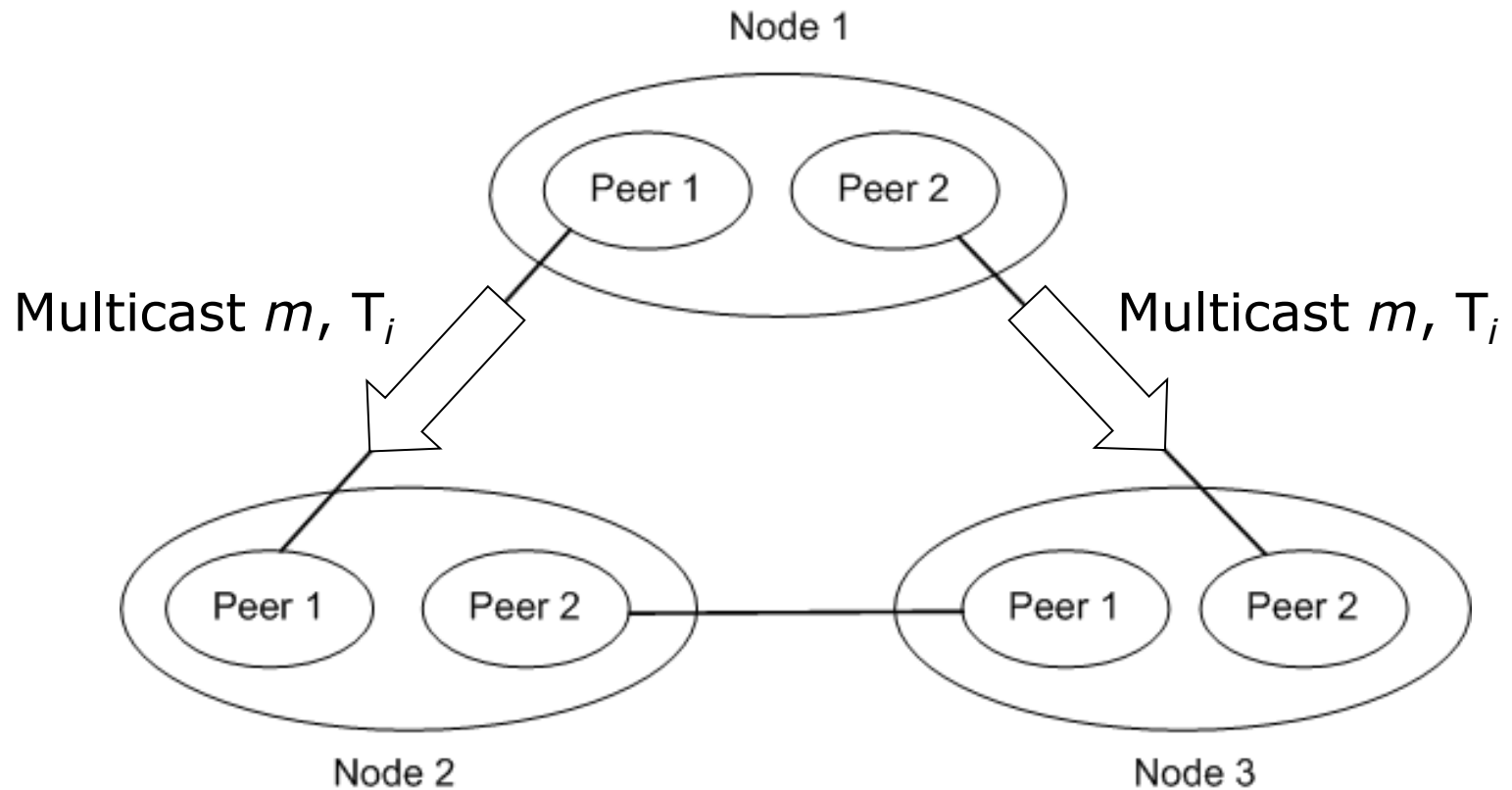
The image shows a screenshot of a code editor window titled "10-peer-twisted-server.py". The editor displays a Python script that uses the `optparse` module for command-line argument parsing. The script imports `Protocol` and `ClientFactory` from `twisted.internet.protocol`, `reactor` from `twisted.internet`, and `time`. It defines a `parse_args()` function that sets up a `parser` with a usage string and a sample command: `python peer.py server 127.0.0.1:port`. The script then checks if the number of arguments is 2; if not, it prints the help and exits. If there are 2 arguments, it assigns `peertype` and `addresses`. A `parse_address(addr)` function is defined to parse the address string, defaulting to `127.0.0.1` if no host is provided, and ensuring the port is a digit. Finally, the script returns `peertype` and the list of `addresses`.

```
11 import optparse
12
13 from twisted.internet.protocol import Protocol, ClientFactory
14 from twisted.internet import reactor
15 import time
16
17 def parse_args():
18     usage = """usage: %prog [options] [client|server] [hostname]:port
19
20     python peer.py server 127.0.0.1:port """
21
22     parser = optparse.OptionParser(usage)
23
24     _, args = parser.parse_args()
25
26     if len(args) != 2:
27         print parser.format_help()
28         parser.exit()
29
30     peertype, addresses = args
31
32     def parse_address(addr):
33         if ':' not in addr:
34             host = '127.0.0.1'
35             port = addr
36         else:
37             host, port = addr.split(':', 1)
38
39         if not port.isdigit():
40             parser.error('Ports must be integers.')
41
42         return host, int(port)
43
44     return peertype, parse_address(addresses)
```

Setting up connections



Multicast & ACK



Multicast & ACK

