Project Presentation

Implement the query-based network monitoring application (from Section 4.5, Exercise 2.)

Chenye Yang cy2540
Bingzhuo Wang bw2632
Zhuoyue Xing zx2269

Requirements

Data generator

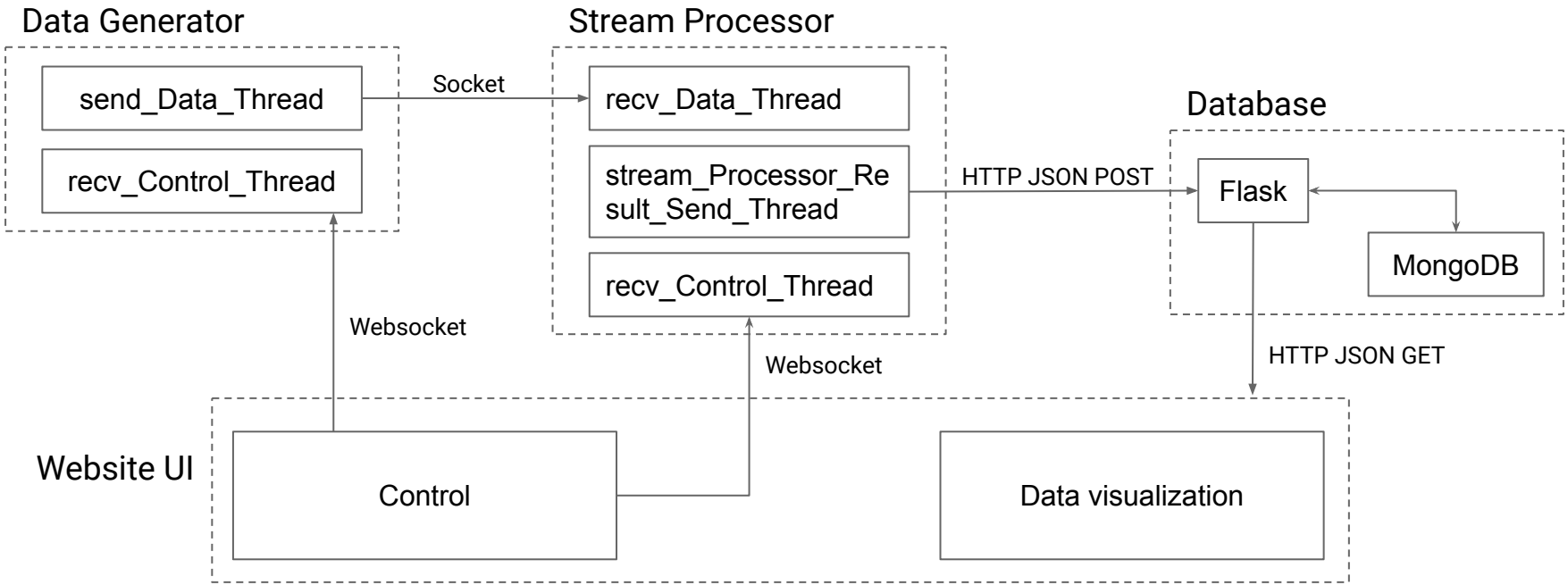
Stream processor

Website UI

Communication between data generator and stream processor

Control of data generator and stream processor through website UI

System Architecture



Data generator

```python
# data generator
class Data_Generator():
    def __init__(self, rate, ipNum, protocolNum, ipPercent, protocolPercent):
        self.rate = rate # <int> Hz
        self.ipNum = ipNum # <int> less than or equal to 15
        self.protocolNum = protocolNum # <int> less than or equal to 19
        # what if the percent list is wrong
        if len(ipPercent) != ipNum:
            self.ipPercentPDF = [1 / ipNum for i in range(ipNum)]
        else:
            self.ipPercentPDF = ipPercent
        if len(protocolPercent) != protocolNum:
            self.protocolPercentPDF = [1 / protocolNum for i in range(protocolNum)]
        else:
            self.protocolPercentPDF = protocolPercent
        # convert the pdf list to cdf list
        self.ipPercent = self.PDF2CDF(self.ipPercentPDF)
        self.protocolPercent = self.PDF2CDF(self.protocolPercentPDF)
        self.protocols = ['SOAP', 'SSDP', 'TCAP', 'UPnP', 'DHCP', 'DNS', 'HTTP', 'HTTPS', 'NFS', 'POP3', 'SMTP', 'SNMP',
                          'FTP', 'NTP', 'IRC', 'Telnet', 'SSH', 'TFTP', 'ANQP']
        self.ips = ['53.215.218.189', '133.98.231.165', '222.186.237.75', '11.71.50.83', '45.43.227.63',
                    '116.168.68.91', '20.232.17.27', '158.223.93.237', '84.191.253.211', '153.17.103.198',
                    '224.80.117.250', '97.211.109.139', '21.50.108.54', '109.126.189.56', '90.227.18.21']

    def PDF2CDF(self, PDF):...

    def data_generator(self):
        # At its finest level, individual flows are identified and associated with an application protocol,
        # a source and destination IP addresses, as well as a set of network packets and their sizes.

        # decide which protocol to generate
        r = random.random()
        i = 0
        for i in range(self.protocolNum):
            if r < self.protocolPercent[i]:
                break
        protocol = self.protocols[i]
        # decide which source IP to generate
        r = random.random()
        i = 0
        for i in range(self.ipNum):
            if r < self.ipPercent[i]:
                break
        sourceIP = self.ips[i]
        # decide which destination IP to generate
        r = random.random()
        i = 0
        for i in range(self.ipNum):
            if r < self.ipPercent[i]:
                break
        destinationIP = self.ips[i]
        packetSize = random.randint(16, 12288)  # generate the size of network packet, Byte
        time = datetime.now(tz=pytz.timezone('US/Eastern')).strftime('%Y-%m-%d_%H:%M:%S.%f')
        # time = datetime.now(tz=pytz.timezone('US/Eastern'))
        dataToSend = '{} {} {} {} {}'.format(time, protocol, sourceIP, destinationIP, packetSize)
        return dataToSend
```

Randomly generate

Predefined

```python
# data sending thread of data generator
class Send_Data_Thread(threading.Thread):
    def __init__(self, threadID, name, rate, ipNum, protocolNum, ipPercent, protocolPercent):...

    def send_data(self):
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        '''
        First attempt:
        data_generator: socket client, sending data
        stream processor: socket server, receiving data
        '''
        ...
        '''
        Second attempt:
        data_generator: socket server, wait for connection from spark streaming
        stream processor: spark streaming, initiative connect to socket server
        '''
        sock.bind(('localhost', 12301))
        sock.listen(5)
        print('{}{}GOOD:{}{} Connection complete. Data Generator is listening for spark stream, from port 12301.'.format(
            Color.GREEN, Color.BOLD, Color.END, Color.END))
        connection, address = sock.accept()
        data_Generator = Data_Generator(self.rate, self.ipNum, self.protocolNum, self.ipPercent, self.protocolPercent)
        try:
            while True:
                time.sleep(1 / self.rate)  # sending frequency
                data = '{}{}'.format(data_Generator.data_generator(), '\n').encode('utf-8')
                connection.send(data)
                global stop_send_Thread
                if stop_send_Thread:
                    stop_send_Thread = False # reset this flag so that another send_Thread can start
                    sock.close() # close socket before exit
                    return # break the loop and then this thread is terminated
        except socket.error:
            print('Connection is closed by a peer. Please manually restart the data generator.')
            sock.close()

    def run(self) -> None:
        # override run() in Thread. When start() is called, run() is called
        self.send_data()
```

Send generated data

Socket

```python
# control receiving thread of data generator
class Recv_Control_Thread(threading.Thread):
    def __init__(self, threadID, name):
        threading.Thread.__init__(self)
        self.threadID = threadID
        self.name = name

    def recv_control(self):
        '''using socket to communicate with web ui'''
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.bind(('localhost', 12302))  # port localhost:12302 is used to receive control signal
        sock.listen(5)  # the max connection number, FIFO
        print('{}{}GOOD:{}{} Connection complete. Data Generator is listening for control signal, from port 12302.'.format(
            Color.GREEN, Color.BOLD, Color.END, Color.END))
        while True:  # wait for connection
            connection, address = sock.accept()
            control = connection.recv(1024).decode('utf-8')
            if control == 'start':  # control receiving thread of data generator
                connect
                global
                stop_se
            elif contro
                connect
                global
                global
                global
                global
                global
                send_Da

                send_Da
                print('
                send_Da
            connection.
            connection.
            print('The

    def run(self) -> No
        # override run(
        self.recv_contr
```

```python
def recv_control():
    async def generator(websocket, path):
        '''using websocket to communicate with web ui'''
        global rate
        global ipNum
        global protocolNum
        global ipPercent
        global protocolPercent
        global stop_send_Thread
        print('{}{}GOOD:{}{} Control receiving thread started'.format(Color.GREEN, Color.BOLD, Color.END, Color.END))
        msg = await websocket.recv()
        print(msg)
        response = 'RCVD'
        if msg == 'stop':
            # response = 'stop signal received, closing now'
            stop_send_Thread = True
        elif msg == 'start':
            # response = 'start signal received, starting now'
            send_Data_Thread = Send_Data_Thread(threadID=1, name='send_Data_Thread', rate=rate, ipNum=ipNum,
                                               protocolNum=protocolNum, ipPercent=ipPercent,
                                               protocolPercent=protocolPercent)
            send_Data_Thread.start()
            print('{}{}GOOD:{}{} Data sending thread started'.format(Color.GREEN, Color.BOLD, Color.END, Color.END))
            send_Data_Thread.join()
        elif msg.split('_')[0] == 'change':
            stop_send_Thread = True
            time.sleep(0.5) # wait for the thread ending
            msgs = msg.split('_')
            rate = int(msgs[1])
            ipNum = int(msgs[2])
            protocolNum = int(msgs[3])
            ipPercent = list(map(lambda x: float(x), msgs[4][1:-1].split(',')))
            protocolPercent = list(map(lambda x: float(x), msgs[5][1:-1].split(',')))
            send_Data_Thread = Send_Data_Thread(threadID=1, name='send_Data_Thread', rate=rate, ipNum=ipNum,
                                               protocolNum=protocolNum, ipPercent=ipPercent,
                                               protocolPercent=protocolPercent)
            send_Data_Thread.start()
            print('{}{}GOOD:{}{} Data sending thread started'.format(Color.GREEN, Color.BOLD, Color.END, Color.END))
            send_Data_Thread.join()
        await websocket.send(response)
    start_server = websockets.serve(generator, "localhost", 12302) # port localhost:12302 is used to receive control sig
    asyncio.get_event_loop().run_until_complete(start_server)
    asyncio.get_event_loop().run_forever()
```

Websocket

Stream processor

Process

```python
# stream processor
class Stream_Processor_Thread(threading.Thread):
    def __init__(self, threadID, name, H, T, k, X):...

    def change_parameter(self, H, T, k, X):...

    # list protocols that are consuming more than H percent of the total external
    # bandwidth over the last T time units
    def function1(self):...

    # List the top-k most resource intensive protocols over the last T time units
    def function2(self):
        log_simple = self.log_tuples.map(lambda x: (x[1], x[4])) # ('protocol', packet_size)
        log_agg = log_simple.reduceByKey(lambda x, y: x + y) # sum up the packet sizes
        log_sorted = log_agg.transform(lambda rdd: rdd.sortBy(lambda x: x[1], ascending=False)) # sort, top-k
        print('the top-{} most resource intensive protocols over the last {} time units'.format(self.k, self.T))
        log_sorted.pprint(num=self.k)
        # log_top_k = log_sorted.take(self.k)
        # print(['  '.join(map(str, item)) for item in log_top_k])
        return

    # List all protocols that are consuming more than X times the standard deviation of
    # the average traffic consumption of all protocols over the last T time units
    def function3(self):...

    # List IP addresses that are consuming more than H percent of the total external
    # bandwidth over the last T time units
    def function4(self):...

    # List the top-k most resource intensive IP addresses over the last T time units
    def function5(self):...

    # List all IP addresses that are consuming more than X times the standard deviation
    # of the average traffic consumption of all IP addresses over the last T time units
    def function6(self):...

    # override run() in Thread. When start() is called, run() is called.
    def run(self) -> None:...
```

StreamingContext listen to port

```python
        self.conf = pyspark.SparkConf().setAppName('Project').setMaster('local[*]')  # set the configuration
        self.sc = pyspark.SparkContext(conf=self.conf)  # creat a spark context object
        self.sc.setLogLevel("ERROR")
        self.ssc = StreamingContext(self.sc, self.T)  # take all data received in T second
        # self.ssc.checkpoint('/Users/yangchenye/Downloads/spark_checkpoint')
        self.log_lines = self.ssc.socketTextStream('localhost', 12301)
        # (datetime, protocol, source IP, destination IP, packet size)
        self.log_tuples = self.log_lines.map(lambda x: (
            datetime.strptime(x.split(' ')[0], '%Y-%m-%d_%H:%M:%S.%f'), x.split(' ')[1], x.split(' ')[2],
            x.split(' ')[3], int(x.split(' ')[4])))
```

Socket

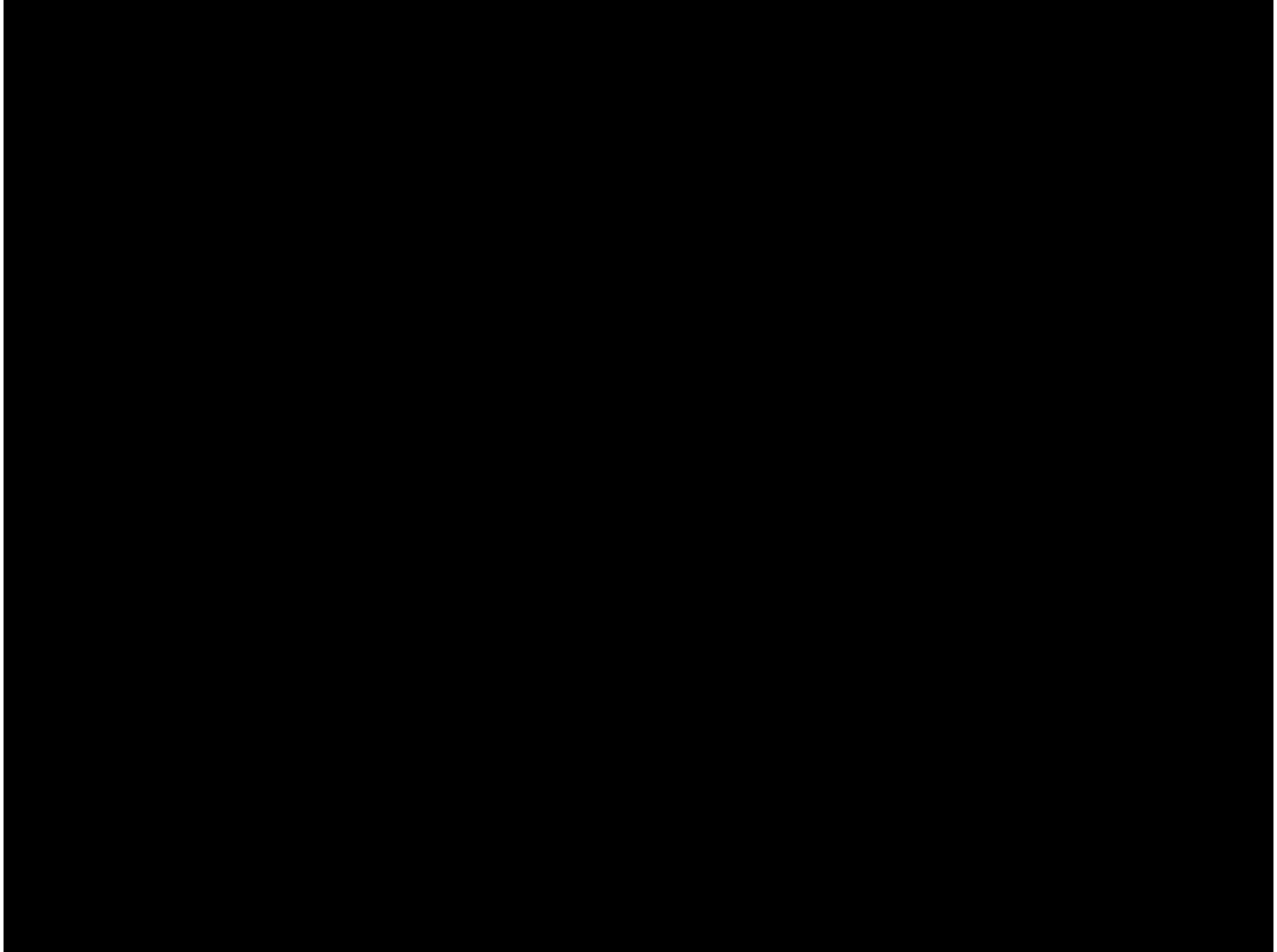
```python
# control receiving thread of stream processor
class Recv_Control_Thread(threading.Thread):
    def __init__(self, threadID, name):
        threading.Thread.__init__(self)
        self.threadID = threadID
        self.name = name

    def recv_control(self):
        '''using socket to communicate with web ui'''
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.bind(('localhost', 12303))  # port localhost:12303 is used to receive contr
        sock.listen(5)  # the max connection number, FIFO
        print(
            '{}{}GOOD:{}{} Connection complete. Stream Processor is listening control si
                Color.GREEN, Color.BOLD, Color.END, Color.END))
        while True:  # wait for connection
            connection, address = sock.accept()
            control = connection.recv(1024).decode("utf-8")
            if control == 'stop':
                connection.send(b'Stop receiving data signal received, closing now')
                global stop_receive_Thread
                stop_receive_Thread = True
            # elif control == 'start_send_Thread':
            #     connection.send(b'Start signal received, starting now')
            #     send_Thread = Send_Thread(threadID=1, name='send_Thread')
            #     send_Thread.start()
            #     send_Thread.join()
            connection.send(b'Control signal received')
            connection.close()
            print('The control signal received is: ' + control)

    # override run() in Thread. When start() is called, run() is called.
    def run(self) -> None:
        self.recv_control()
```

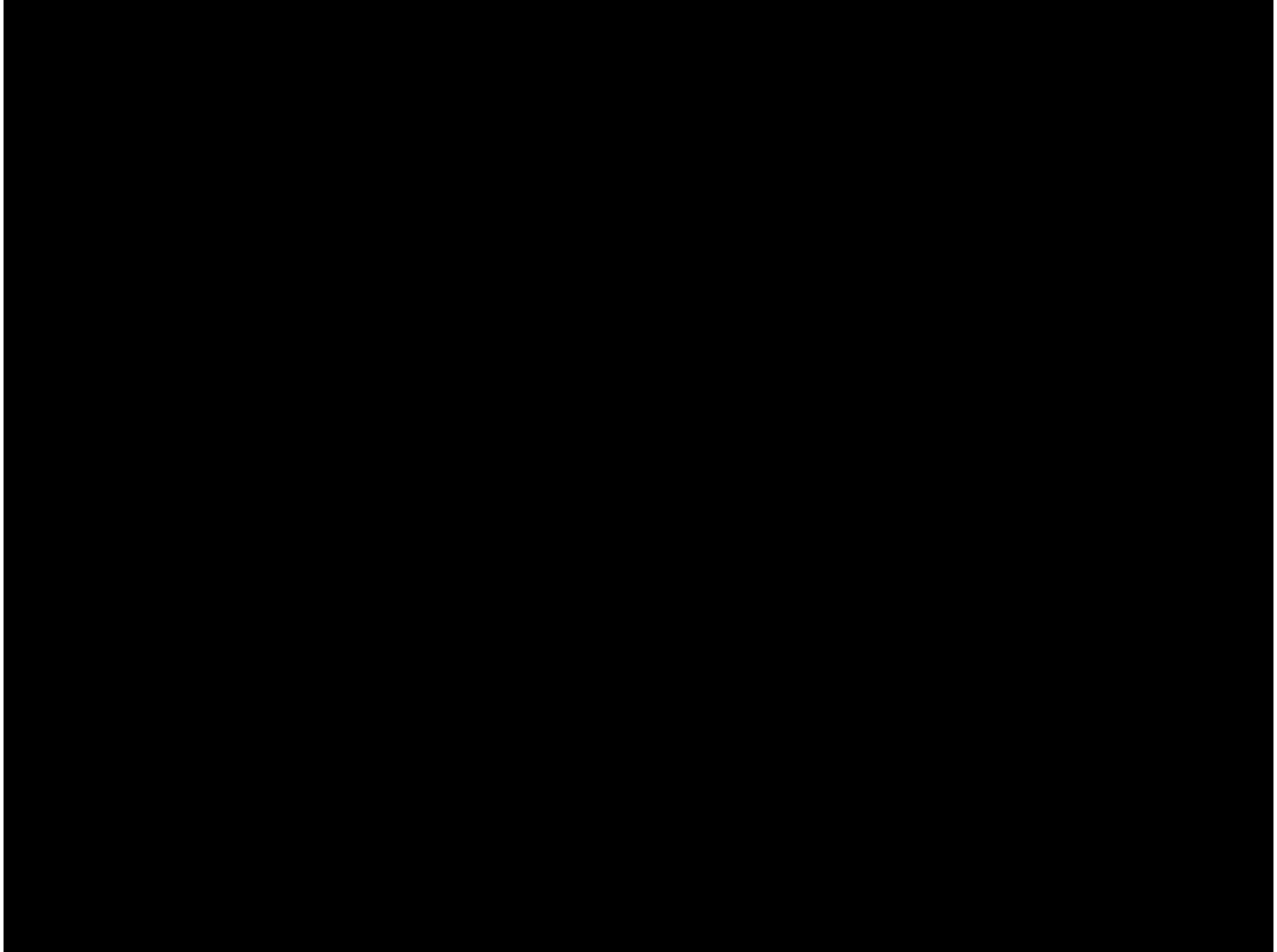
Websocket

```python
# control receiving thread of stream processor
def recv_control():
    '''using websocket to communicate with web ui'''
    async def generator(websocket, path):
        global H
        global T
        global k
        global X
        global stop_receive_Thread
        print('{}{}GOOD:{}{} Control receiving thread started'.format(Color.GREEN, Color.BOLD, Color.END, Color.END))
        msg = await websocket.recv()
        print(msg)
        response = 'RCVD'
        if msg == 'stop':
            # response = 'stop signal received, closing now'
            stop_receive_Thread = True
        elif msg == 'start':
            # response = 'start signal received, starting now'
            stream_Processor_Thread = Stream_Processor_Thread(threadID=3, name='stream_Processor_Thread', H=H, T=T, k=k,
                                                              X=X)
            stream_Processor_Thread.start()
            print('{}{}GOOD:{}{} Stream processor thread started'.format(Color.GREEN, Color.BOLD, Color.END, Color.END))
            stream_Processor_Thread.join()
        elif msg.split('_')[0] == 'change':
            stop_receive_Thread = True
            time.sleep(0.5)  # wait for the thread ending
            msgs = msg.split('_')
            H, T, k, X = float(msgs[1]), int(msgs[2]), int(msgs[3]), float(msgs[4])
            stream_Processor_Thread = Stream_Processor_Thread(threadID=3, name='stream_Processor_Thread', H=H, T=T, k=k,
                                                              X=X)
            stream_Processor_Thread.start()
            print('{}{}GOOD:{}{} Stream processor thread started'.format(Color.GREEN, Color.BOLD, Color.END, Color.END))
            stream_Processor_Thread.join()

    start_server = websockets.serve(generator, "localhost", 12303) # port localhost:12303 is used to receive control sign

    asyncio.get_event_loop().run_until_complete(start_server)
    asyncio.get_event_loop().run_forever()
```

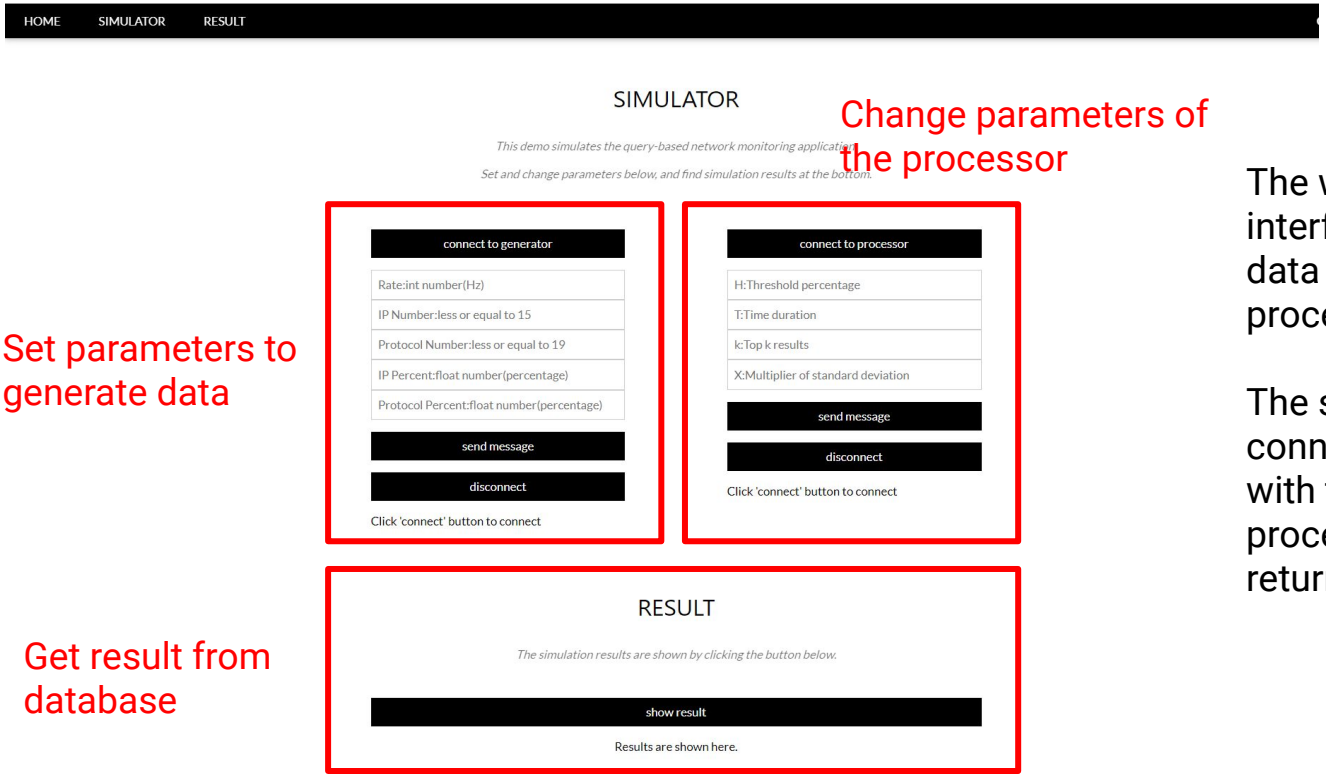
HTTP JSON POST

```python
#url here is the url that flask is running on(http://127.0.0.1:5000/) add "6889final/insert"
def data_sender(sdata,url= "http://127.0.0.1:5000/6889final/insert"):
    headers = {
        "Content-Type": "application/json; charset=UTF-8"
    }
    response = requests.post(url, data=json.dumps(sdata), headers=headers).text
    #if data is json type,change the above line to
    #response = requests.post(url, data=sdata, headers=headers).text
    print(response)
```





Website UI



Change parameters of the processor

Set parameters to generate data

Get result from database

The website performs as the interface to interact with database, data generator and streaming processor.

The simulator part use socket connection to change parameters with the generator and the processor, the result part will return result from database.

Communications and control

Communication between data generator and streaming processor: Socket programming

Use client-server socket to send data from data generator to streaming processor.

Control between website UI and data generator/streaming processor: Websocket

Use Websocket to send message from website UI to data generator and streaming processor.

Database

We built our database on MongoDB database program and use Python Flask frame to connect database with Stream Processor part and Website UI. In our Flask program, we implemented the following functions using HTTP requests.

1.Get all data from a table in the database
2.Get the data that is most recently inserted into a table
3.Insert the data from Stream Processor to a table in the database
4.Delete all the data in a table in the database

Database

```python
@app.route('/6889final/insert', methods=['POST'])
def insert_by_sensornumber():
    result = None
    try:
        if request.method == 'POST':
            data = None
            try:
                if request.data is not None:
                    data = request.json
                else:
                    data = None
            except Exception as e:
                # This would fail the request in a more real solution.
                data = "You sent something but I could not get JSON out of it."
            location = mongo.db.E6889final
            location.insert_one(data)
            res = "Insert successfully"
            rsp = Response(json.dumps(res), status=200, content_type="application/json")
            return rsp
        else:
            result = "Insert data get from Stream Processor"
            return result, 400, {'Content-Type': 'text/plain; charset=utf-8'}
    except Exception as e:
        print(e)
        return handle_error(e, result)
```