In [10]:
```
import numpy as np
import pandas as pd
import re
import math
from sklearn.decomposition import PCA, FactorAnalysis
from sklearn.preprocessing import StandardScaler
from sklearn import svm
import matplotlib.pyplot as plt
import matplotlib
import os
%matplotlib inline


def plot_confusion_matrix(y_true, y_pred, classes,
                          normalize=False,
                          title=None,
                          cmap=plt.cm.Blues, font_size=10, fig_size=(12,10)):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if not title:
        if normalize:
            title = 'Normalized confusion matrix'
        else:
            title = 'Confusion matrix, without normalization'

    # Compute confusion matrix
    cm = confusion_matrix(y_true, y_pred)
    # Only use the labels that appear in the data
    #classes = classes[unique_labels(y_true, y_pred)]
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        #print("Normalized confusion matrix")
    #else:
    #    print('Confusion matrix, without normalization')

    #print(cm)
    plt.rcParams.update({'font.size': font_size})
    fig, ax = plt.subplots(figsize=fig_size, dpi= 80, facecolor='w', edgecolo
r='k')
    im = ax.imshow(cm, interpolation='nearest', cmap=cmap)
    ax.figure.colorbar(im, ax=ax)
    # We want to show all ticks...
    ax.set(xticks=np.arange(cm.shape[1]),
           yticks=np.arange(cm.shape[0]),
           # ... and label them with the respective list entries
           xticklabels=classes, yticklabels=classes,
           title=title,
           ylabel='True label',
           xlabel='Predicted label')

    # Rotate the tick labels and set their alignment.
    plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
             rotation_mode="anchor")

    # Loop over data dimensions and create text annotations.
    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
```

Load and Trim Datasets

Loads data and trims tuples down to correct 2 hour time period

In [11]:
```
def load_dataset(path):
    out_data = pd.DataFrame()
    for sub_dir in os.listdir(path):
        temp_path = os.path.join(path, sub_dir)
        temp_path = os.path.join(temp_path, "log_tcp_complete")
#        print(temp_path)
        if os.path.isfile(temp_path):
#            print('1')
#            temp_data = read_in_file(temp_path)
            temp_data = pd.read_csv(temp_path, delimiter= '\s', index_col=Fals
e)
            temp_data.shape
            out_data = out_data.append(temp_data)
        #np.concatenate((out_data, temp_data))
#    return
    df = out_data
    #load data
#def trim_dataset(df):

    #create timestamp
    df['t_first']=df['first:29'].astype('float').astype("datetime64[ms]")
    df['t_last']=df['last:30'].astype('float').astype("datetime64[ms]")

    #find start/end time
    t_start = df.t_first.min()
    t_end = t_start+ pd.Timedelta(hours=2)
    t_max = df.t_last.max()

    #trim data after test complete (tstat ran too long in bash shell)
    df = df[(df['t_first'] >= t_start) & (df['t_last'] < t_end)]
#    df = df.values

    df.drop(columns=['t_first', 't_last'])
    df.drop(df.index[0])

#    csv = get_data_row(df.to_csv( index = False, sep = " "))

    return df
```

Load Label Indexes

This chunk loads a file that contains the labels we want to load from the datasets as well as their indicies.

In [12]:
```
infile = open("tstat_labels_indexes.txt" ,'r')
data_field_list = []
for line in infile.readlines():
    if ":" in line:
        data_field = str(re.search('%s(.*)%s' % ("\"", "\""), line).group(1))
        index = int(re.search('%s(.*)%s' % (":", ","), line).group(1))
        data_field_list.append((data_field, index))

index_to_key_dict = {}
key_to_index_dict = {}
data_field_labels = []
for data_field, index in data_field_list:
    key_to_index_dict[data_field] = index
    index_to_key_dict[index] = data_field
    data_field_labels.append(data_field)

len(data_field_list)
```

Out[12]: 89

Read in a dataset file

In [13]:
```
def read_in_file(input):
    entries = []
    labels = None
    for i, line in enumerate(input):
        row = get_data_row(line)
        row = clean_data_row(row)
        if row != []:
            entries.append(row)
    entries = np.array(entries)
    return entries
```

Get data row

Called by the read in file function. Loads a single line from the dataset files. Super inefficient, but only loads labels which are in the data field list.

In [14]:
```
def get_data_row(line):
    global index_to_key_dict
#    print(line)
    row = []
    labels = []
    c_pkt_cnt = 0
    s_pkt_cnt = 0
    c_bytes_cnt = 0
    s_bytes_cnt = 0
    for data_field, index in data_field_list:
        if data_field == "client_pkt_cnt":
            try:
                c_pkt_cnt = line[index]
                c_pkt_cnt = max(float(c_pkt_cnt), 1)
            except:
                c_pkt_cnt = 1
            #if c_pkt_cnt < 32:
            #    return []
        elif data_field == "serv_pkt_cnt":
            try:
                s_pkt_cnt = line[index]
                s_pkt_cnt = max(float(s_pkt_cnt), 1)
            except:
                s_pkt_cnt = 1
        elif data_field == "client_bytes_cnt":
            try:
                c_bytes_cnt = line[index]
                c_bytes_cnt = max(float(c_bytes_cnt), 1)
            except:
                c_bytes_cnt = 1
        elif data_field == "serv_bytes_cnt":
            try:
                s_bytes_cnt = line[index]
                s_bytes_cnt = max(float(s_bytes_cnt), 1)
            except:
                s_bytes_cnt = 1

    for data_field, index in data_field_list:
        try:
            val = line[index]
            val = float(val)
        except:
            val = 0
        if data_field in ["client_pkt_cnt", "client_rst_cnt", "client_ack_cnt",
"client_pkt_data", "client_pkt_retx",
                         "client_syn_cnt", "client_fin_cnt", "client_pkt_ret
x"]:
            val /= c_pkt_cnt
        elif data_field in ["client_bytes_uniq", "client_bytes_cnt", "client_by
tes_retx"]:
            val /= c_bytes_cnt
        elif data_field in ["serv_pkt_cnt", "serv_rst_cnt", "serv_ack_cnt", "se
rv_ack_pck_cnt", "serv_pkts_data",
                            "serv_pkts_retx", "serv_syn_cnt", "serv_fin_cnt"]:
            val /= s_pkt_cnt
        elif data_field in ["serv_bytes_uniq", "serv_btyes_cnt", "serv_pkts_ret
x"]:
            val /= s_bytes_cnt
        row.append(val)
    return row
```

Clean data row

Not implemented

In [15]:
```
def clean_data_row(in_row):
    global index_to_key_dict, key_to_index_dict

    for data_field, index in data_field_list:
        if math.isnan(in_row[index]):
            try:
                in_row[index] == 0

            except:
                print('err')



    return in_row
```

Get dataset

Loads all files from a directory

In [16]:
```
def get_dataset(path):
    print(path)
    temp_data = load_dataset(path)
    temp_data = temp_data.drop(['t_first', 't_last'], axis=1)
#csv = (temp_data.to_csv( index = False, sep = " "))
#entries = []
#lines = csv.splitlines()

    entries = []

    temp_data = temp_data.values
    for row in temp_data:
        myRow = get_data_row(row)
        myRow= [float(i) for i in myRow]

#         myRow = clean_data_row(myRow)
        entries.append(myRow)

    entries = np.array(entries)

    return entries

    # add pandas to list and row algo

#pd.DataFrame(temp_data)
#print("output::", csv)
```

In [17]:
```
temp_data = get_dataset("./normal")
```

```
./normal

/home/dave/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:11: Pars
erWarning: Falling back to the 'python' engine because the 'c' engine does not
support regex separators (separators > 1 char and different from '\s+' are inte
rpreted as regex); you can avoid this warning by specifying engine='python'.
  # This is added back by InteractiveShellApp.init_path()
```

Load all datasets

Load all datasets Create numerical lables for each class, and a different set of labels for each subclass.

In [18]:
```
normal = get_dataset("./normal")
corr_01 = get_dataset("./corrupt_0.1perc")
corr_05 = get_dataset("./corrupt_0.5perc")
corr_10 = get_dataset("./corrupt_1.0perc")
delay_1_1 = get_dataset("./delay_1_var_1")
delay_5_2 = get_dataset("./delay_5_var_2")
delay_10_5 = get_dataset("./delay_10_var_5")
delay_25_20 = get_dataset("./delay_25_var_20")
drop_01 = get_dataset("./drop_01_perc")
drop_001 = get_dataset("./drop_001_perc")
drop_0005 = get_dataset("./drop_0005_perc")
dup_1 = get_dataset("./dup-1-p")
dup_2 = get_dataset("./dup_2perc")

normal = np.nan_to_num(normal)
corr_01 = np.nan_to_num(corr_01)
corr_05 =  np.nan_to_num(corr_05)
corr_10 = np.nan_to_num(corr_10)
delay_1_1 = np.nan_to_num(delay_1_1)
delay_5_2 = np.nan_to_num(delay_5_2)
delay_10_5 = np.nan_to_num(delay_10_5)
delay_25_20 = np.nan_to_num(delay_25_20)
drop_01 = np.nan_to_num(drop_01)
drop_001 = np.nan_to_num(drop_001)
drop_0005 = np.nan_to_num(drop_0005)
dup_1 = np.nan_to_num(dup_1)
dup_2 = np.nan_to_num(dup_2)
```

```
./normal

/home/dave/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:11: Pars
erWarning: Falling back to the 'python' engine because the 'c' engine does not
support regex separators (separators > 1 char and different from '\s+' are inte
rpreted as regex); you can avoid this warning by specifying engine='python'.
  # This is added back by InteractiveShellApp.init_path()

./corrupt_0.1perc
./corrupt_0.5perc
./corrupt_1.0perc
./delay_1_var_1
./delay_5_var_2
./delay_10_var_5
./delay_25_var_20
./dup-1-p
./dup_2perc
```

In [19]:
```
normal
```

Out[19]:
```
array([[1.00000000e+00, 0.00000000e+00, 8.00000000e-01, ...,
        0.00000000e+00, 1.00000000e+00, 1.00000000e+00],
       [1.00000000e+00, 0.00000000e+00, 8.88888889e-01, ...,
        0.00000000e+00, 2.00000000e+00, 3.00000000e+00],
       [1.00000000e+00, 0.00000000e+00, 9.99901439e-01, ...,
        0.00000000e+00, 9.91000000e+02, 0.00000000e+00],
       ...,
       [1.00000000e+00, 0.00000000e+00, 8.88888889e-01, ...,
        0.00000000e+00, 2.00000000e+00, 3.00000000e+00],
       [1.00000000e+00, 0.00000000e+00, 9.99902487e-01, ...,
        0.00000000e+00, 6.20000000e+02, 0.00000000e+00],
       [1.00000000e+00, 0.00000000e+00, 9.37500000e-01, ...,
        0.00000000e+00, 7.00000000e+00, 6.00000000e+00]])
```

In [20]:
```
all_data = np.concatenate((normal,
                           corr_01, corr_05, corr_10,
                           delay_1_1, delay_5_2,delay_10_5,delay_25_20,
                           drop_01, drop_001,
                           dup_1, dup_2))

pd.DataFrame(all_data)


#all_data = StandardScaler().fit_transform(all_data)
```

Out[20]:

	0	1	2	3	4	5	6	7	8	9	...	79	80	81	8
0	1.0	0.0	0.800000	2.0	1.000000	0.200000	1.0	0.000000	0.000000e+00	0.200000	...	0.0	0.0	0.0	0
1	1.0	0.0	0.888889	5.0	1.000000	0.222222	1.0	0.000000	0.000000e+00	0.111111	...	0.0	0.0	0.0	0
2	1.0	0.0	0.999901	1.0	1.000000	0.999803	1.0	0.000000	0.000000e+00	0.000099	...	0.0	0.0	0.0	0
3	1.0	0.0	0.937500	7.0	1.000000	0.437500	1.0	0.000000	0.000000e+00	0.062500	...	0.0	0.0	0.0	0
4	1.0	0.0	0.800000	2.0	1.000000	0.200000	1.0	0.000000	0.000000e+00	0.200000	...	0.0	0.0	0.0	0
5	1.0	0.0	0.888889	5.0	1.000000	0.222222	1.0	0.000000	0.000000e+00	0.111111	...	0.0	0.0	0.0	0
6	1.0	0.0	0.999904	1.0	1.000000	0.999808	1.0	0.000096	1.479939e-09	0.000096	...	0.0	0.0	0.0	0
7	1.0	0.0	0.937500	7.0	1.000000	0.437500	1.0	0.000000	0.000000e+00	0.062500	...	0.0	0.0	0.0	0
8	1.0	0.0	0.800000	2.0	1.000000	0.200000	1.0	0.000000	0.000000e+00	0.200000	...	0.0	0.0	0.0	0
9	1.0	0.0	0.888889	5.0	1.000000	0.222222	1.0	0.000000	0.000000e+00	0.111111	...	0.0	0.0	0.0	0
10	1.0	0.0	0.999905	1.0	1.000000	0.999810	1.0	0.000000	0.000000e+00	0.000095	...	0.0	0.0	0.0	0
11	1.0	0.0	0.937500	7.0	1.000000	0.437500	1.0	0.000000	0.000000e+00	0.062500	...	0.0	0.0	0.0	0
12	1.0	0.0	0.800000	2.0	1.000000	0.200000	1.0	0.000000	0.000000e+00	0.200000	...	0.0	0.0	0.0	0
13	1.0	0.0	0.888889	5.0	1.000000	0.222222	1.0	0.000000	0.000000e+00	0.111111	...	0.0	0.0	0.0	0
14	1.0	0.0	0.999904	1.0	1.000000	0.999807	1.0	0.000096	1.489406e-09	0.000096	...	0.0	0.0	0.0	0
15	1.0	0.0	0.937500	7.0	1.000000	0.437500	1.0	0.000000	0.000000e+00	0.062500	...	0.0	0.0	0.0	0
16	1.0	0.0	0.800000	2.0	1.000000	0.200000	1.0	0.000000	0.000000e+00	0.200000	...	0.0	0.0	0.0	0
17	1.0	0.0	0.888889	5.0	1.000000	0.222222	1.0	0.000000	0.000000e+00	0.111111	...	0.0	0.0	0.0	0
18	1.0	0.0	0.999904	1.0	1.000000	0.999808	1.0	0.000000	0.000000e+00	0.000096	...	0.0	0.0	0.0	0
19	1.0	0.0	0.937500	7.0	1.000000	0.437500	1.0	0.000000	0.000000e+00	0.062500	...	0.0	0.0	0.0	0
20	1.0	0.0	0.800000	2.0	1.000000	0.200000	1.0	0.000000	0.000000e+00	0.200000	...	0.0	0.0	0.0	0
21	1.0	0.0	0.888889	5.0	1.000000	0.222222	1.0	0.000000	0.000000e+00	0.111111	...	0.0	0.0	0.0	0
22	1.0	0.0	0.999905	1.0	1.000000	0.999809	1.0	0.000000	0.000000e+00	0.000095	...	0.0	0.0	0.0	0
23	1.0	0.0	0.937500	7.0	1.000000	0.437500	1.0	0.000000	0.000000e+00	0.062500	...	0.0	0.0	0.0	0
24	1.0	0.0	0.800000	2.0	1.000000	0.200000	1.0	0.000000	0.000000e+00	0.200000	...	0.0	0.0	0.0	0
25	1.0	0.0	0.800000	2.0	1.000000	0.200000	1.0	0.000000	0.000000e+00	0.200000	...	0.0	0.0	0.0	0
26	1.0	0.0	0.800000	2.0	1.000000	0.200000	1.0	0.000000	0.000000e+00	0.200000	...	0.0	0.0	0.0	0
27	1.0	0.0	0.800000	2.0	1.000000	0.200000	1.0	0.000000	0.000000e+00	0.200000	...	0.0	0.0	0.0	0
28	1.0	0.0	0.800000	2.0	1.000000	0.200000	1.0	0.000000	0.000000e+00	0.200000	...	0.0	0.0	0.0	0
29	1.0	0.0	0.888889	5.0	1.000000	0.222222	1.0	0.000000	0.000000e+00	0.111111	...	0.0	0.0	0.0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
27272	1.0	0.0	0.999900	1.0	0.999900	0.999801	1.0	0.000199	1.004185e-04	0.000100	...	0.0	0.0	0.0	0
27273	1.0	0.0	0.937500	7.0	1.000000	0.437500	1.0	0.000000	0.000000e+00	0.062500	...	0.0	0.0	0.0	0
27274	1.0	0.0	0.800000	2.0	1.000000	0.200000	1.0	0.000000	0.000000e+00	0.200000	...	0.0	0.0	0.0	0
27275	1.0	0.0	0.888889	5.0	1.000000	0.222222	1.0	0.000000	0.000000e+00	0.111111	...	0.0	0.0	0.0	0
27276	1.0	0.0	0.999904	1.0	1.000000	0.999807	1.0	0.000096	1.486175e-09	0.000096	...	0.0	0.0	0.0	0
27277	1.0	0.0	0.937500	7.0	1.000000	0.437500	1.0	0.000000	0.000000e+00	0.062500	...	0.0	0.0	0.0	0
27278	1.0	0.0	0.800000	2.0	1.000000	0.200000	1.0	0.000000	0.000000e+00	0.200000	...	0.0	0.0	0.0	0
27279	1.0	0.0	0.888889	5.0	1.000000	0.222222	1.0	0.000000	0.000000e+00	0.111111	...	0.0	0.0	0.0	0

In [21]:
```
all_data = np.concatenate((normal,
                           corr_01, corr_05, corr_10,
                           delay_1_1, delay_5_2,delay_10_5,delay_25_20,
                           drop_01, drop_001, drop_0005,
                           dup_1, dup_2))
all_data
```

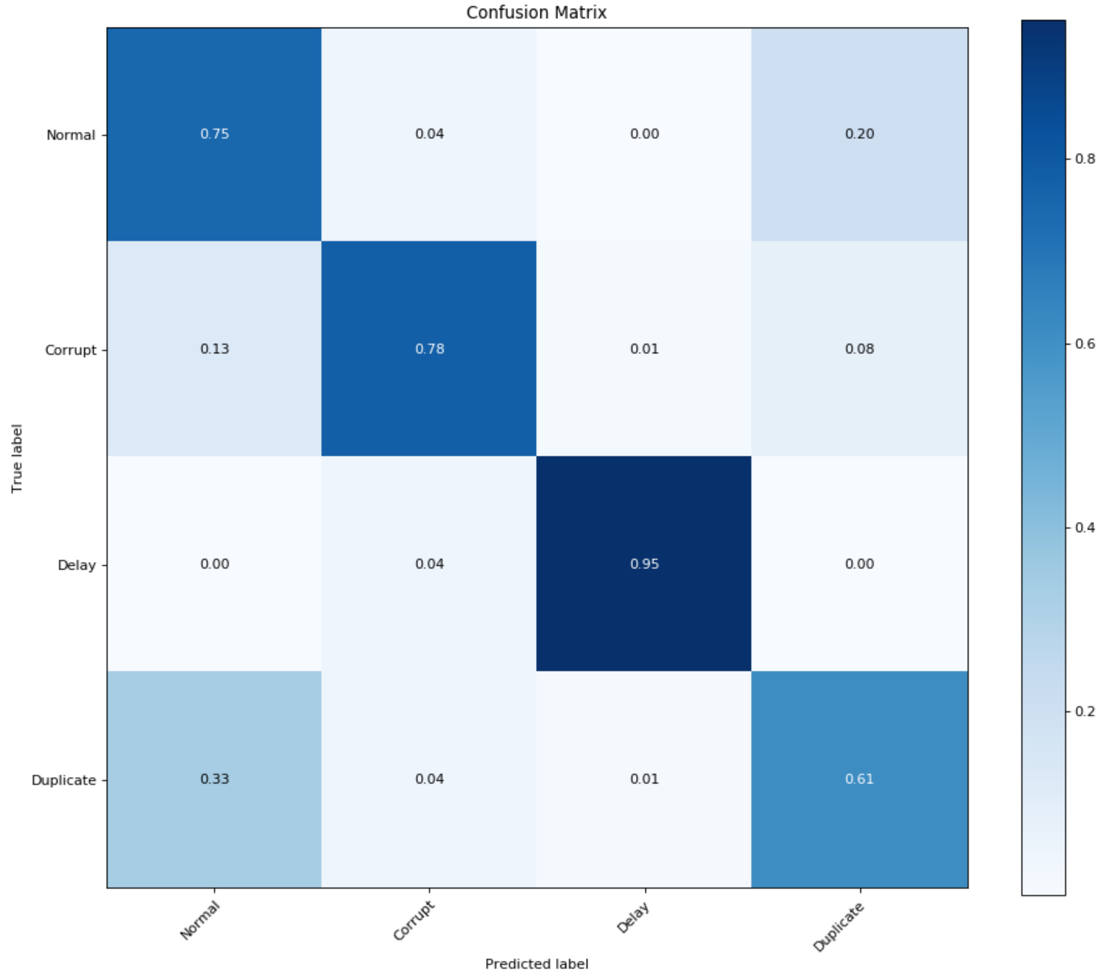
Out[21]:
```
array([[1.00000000e+00, 0.00000000e+00, 8.00000000e-01, ...,
        0.00000000e+00, 1.00000000e+00, 1.00000000e+00],
       [1.00000000e+00, 0.00000000e+00, 8.88888889e-01, ...,
        0.00000000e+00, 2.00000000e+00, 3.00000000e+00],
       [1.00000000e+00, 0.00000000e+00, 9.99901439e-01, ...,
        0.00000000e+00, 9.91000000e+02, 0.00000000e+00],
       ...,
       [1.00000000e+00, 0.00000000e+00, 8.88888889e-01, ...,
        0.00000000e+00, 2.00000000e+00, 3.00000000e+00],
       [1.00000000e+00, 0.00000000e+00, 9.99902449e-01, ...,
        0.00000000e+00, 8.98000000e+02, 0.00000000e+00],
       [1.00000000e+00, 0.00000000e+00, 9.37500000e-01, ...,
        0.00000000e+00, 7.00000000e+00, 6.00000000e+00]])
```

In [22]:
```
all_data = np.nan_to_num(all_data)
all_data = StandardScaler().fit_transform(all_data)
```

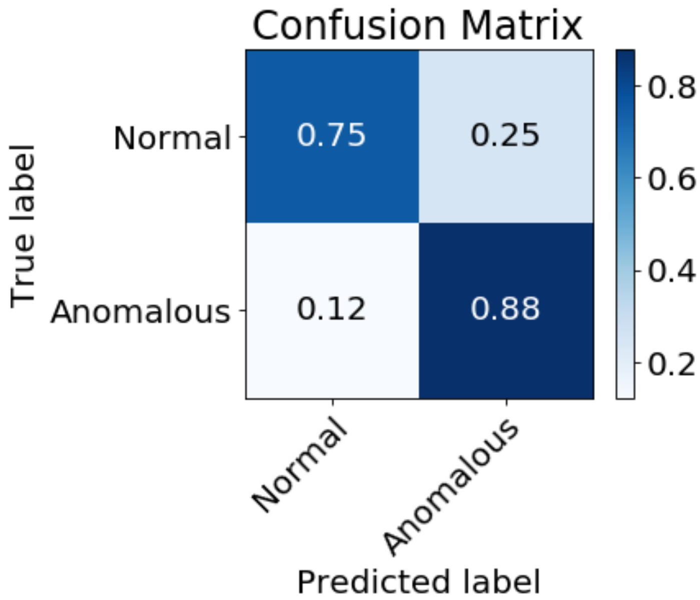
In [23]:
```
a_labels  = np.ones(len(normal))      *1
b_labels  = np.ones(len(corr_01  ))   *2
c_labels  = np.ones(len(corr_05))      *3
d_labels  = np.ones(len(corr_10  ))   *4
e_labels  = np.ones(len(delay_1_1))   *5
f_labels  = np.ones(len(delay_5_2))   *6
g_labels  = np.ones(len(delay_10_5))  *7
h_labels  = np.ones(len(delay_25_20)) *8
i_labels  = np.ones(len(drop_01) )     *9
j_labels  = np.ones(len(drop_001) )    *10
k_labels  = np.ones(len(drop_0005) )   *11
m_labels  = np.ones(len(dup_1))       *13
n_labels  = np.ones(len(dup_2))       *14

data_labels = np.concatenate((a_labels, b_labels, c_labels, d_labels, e_labels,
                              f_labels, g_labels, h_labels,
                              i_labels, j_labels, k_labels,
                              m_labels, n_labels))

a_labels  = np.ones(len(normal  ))    *1
b_labels  = np.ones(len(corr_01  ))   *2
c_labels  = np.ones(len(corr_05  ))   *2
d_labels  = np.ones(len(corr_10  ))   *2
e_labels  = np.ones(len(delay_1_1))   *3
f_labels  = np.ones(len(delay_5_2))   *3
g_labels  = np.ones(len(delay_10_5))  *3
h_labels  = np.ones(len(delay_25_20)) *3
i_labels  = np.ones(len(drop_01))     *4
j_labels  = np.ones(len(drop_001))    *4
k_labels  = np.ones(len(drop_0005))   *4
m_labels  = np.ones(len(dup_1))       *5
n_labels  = np.ones(len(dup_2))       *5

anom_type_data_labels = np.concatenate((a_labels, b_labels, c_labels, d_labels,
e_labels,
                                        f_labels, g_labels, h_labels,
                                        i_labels, j_labels,
                                        m_labels, n_labels))
```

In [24]:
```
from sklearn.model_selection import train_test_split

train_data, test_data, train_labels, test_labels = train_test_split(all_data,
                                                                    anom_type_d
ata_labels, test_size=0.45, random_state=0)
```

In [25]:
```
clf = svm.SVC(kernel='linear', gamma ='auto', max_iter=1000000000, class_weigh
t='balanced')
clf.fit(train_data, train_labels)
predicted_labels = clf.predict(test_data)

from sklearn.metrics import confusion_matrix

class_names = ["Normal", "Corrupt", "Delay", "Duplicate"]

plot_confusion_matrix(test_labels, predicted_labels, normalize=True,classes=cla
ss_names, title='Confusion Matrix')
plt.show()

error_cnt = 0
total_error_cnt = 0
same_class_error = 0
for i in range(len(predicted_labels)):
    if predicted_labels[i] != test_labels[i]:
        total_error_cnt += 1
        # if its normal data
        if predicted_labels[i] == 1 or test_labels[i] == 1:
            error_cnt += 1


print ("Total Error Count : ", total_error_cnt)
print ("Normal Class Error Rate : ", float(error_cnt)/float(len(predicted_label
s)) * 100, "%")
print ("Total Error Rate : ", float(total_error_cnt)/float(len(predicted_label
s)) * 100, "%")


false_pos = 0
false_neg = 0
error_cnt = 0
for i in range(len(predicted_labels)):
    # if there's an error
    if predicted_labels[i] != test_labels[i]:
        error_cnt += 1
        # if we failed to detect anomaly
        if predicted_labels[i] == 1:
            false_neg += 1
        # detected anomaly, but it normal
        else:
            false_pos += 1
print ("Total Errors", error_cnt)
print ("False Positives ", false_pos/ float(len(predicted_labels)) * 100, "%")
print ("False Negatives ", false_neg/ float(len(predicted_labels)) * 100, "%")


shortened_predicted = []
shortened_test_label = []
for i in range(len(predicted_labels)):
    if predicted_labels[i] == 1:
        shortened_predicted.append(1)
    else:
        shortened_predicted.append(2)
    if test_labels[i] == 1:
        shortened_test_label.append(1)
    else:
        shortened_test_label.append(2)

short_class_names = ["Normal", "Anomalous"]
```

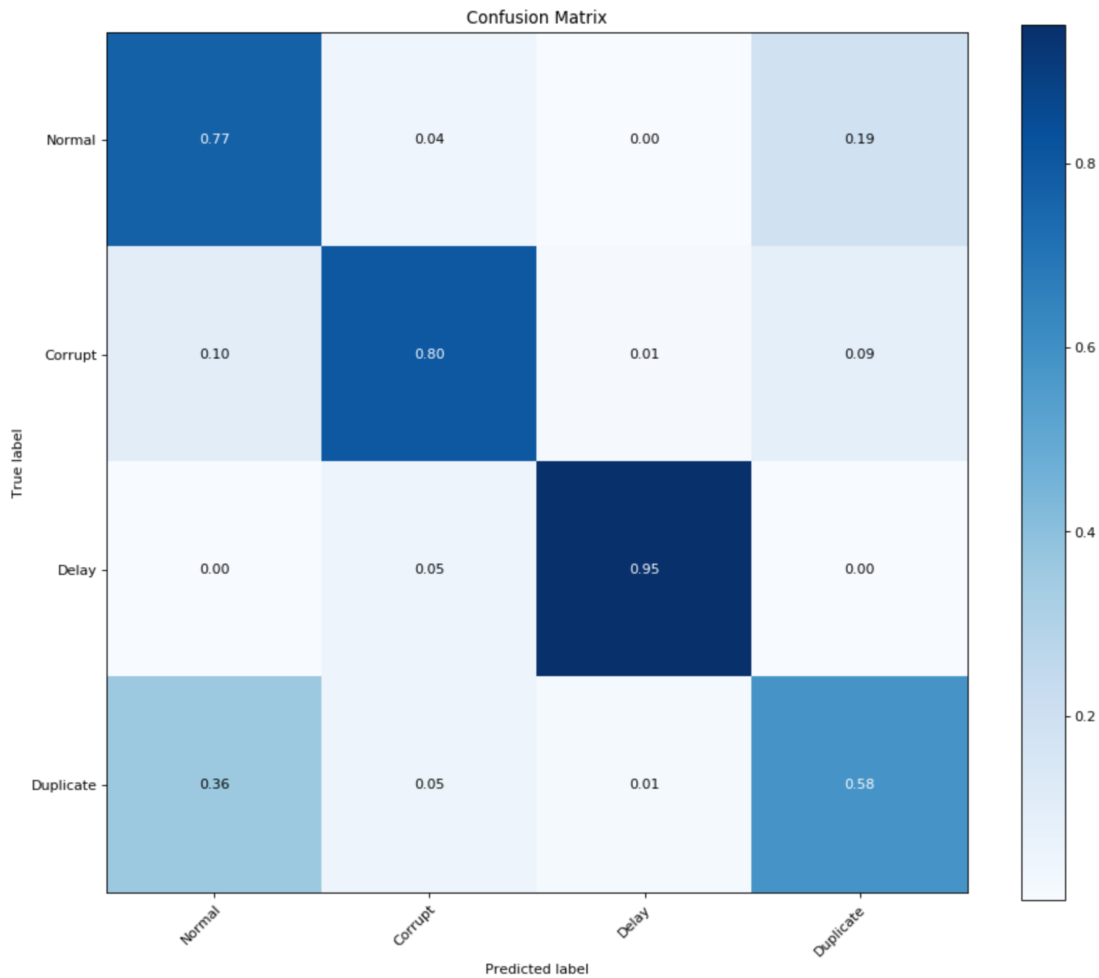



```
Total Error Count :  2390
Normal Class Error Rate :  13.731076021487873 %
Total Error Rate :  19.453035975907536 %
Total Errors 2390
False Positives  8.359107927722611 %
False Negatives  11.093928048184926 %
```

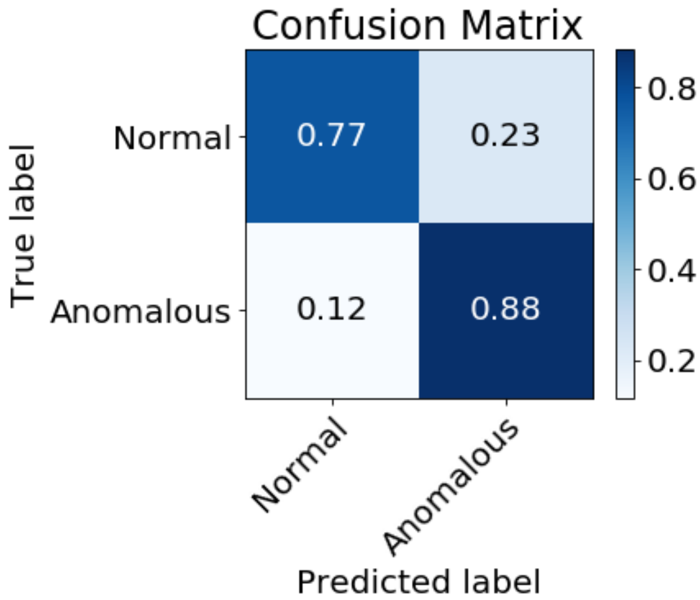


In [26]:
```
from sklearn.model_selection import train_test_split

train_data, test_data, train_labels, test_labels = train_test_split(all_data,
                                                                  anom_type_d
ata_labels, test_size=0.45, random_state=1)
```

In [27]:
```
clf = svm.SVC(kernel='linear', gamma ='auto', max_iter=1000000000, class_weigh
t='balanced')
clf.fit(train_data, train_labels)
predicted_labels = clf.predict(test_data)

from sklearn.metrics import confusion_matrix

class_names = ["Normal", "Corrupt", "Delay", "Duplicate"]

plot_confusion_matrix(test_labels, predicted_labels, normalize=True,classes=cla
ss_names, title='Confusion Matrix')
plt.show()

error_cnt = 0
total_error_cnt = 0
same_class_error = 0
for i in range(len(predicted_labels)):
    if predicted_labels[i] != test_labels[i]:
        total_error_cnt += 1
        # if its normal data
        if predicted_labels[i] == 1 or test_labels[i] == 1:
            error_cnt += 1


print ("Total Error Count : ", total_error_cnt)
print ("Normal Class Error Rate : ", float(error_cnt)/float(len(predicted_label
s)) * 100, "%")
print ("Total Error Rate : ", float(total_error_cnt)/float(len(predicted_label
s)) * 100, "%")


false_pos = 0
false_neg = 0
error_cnt = 0
for i in range(len(predicted_labels)):
    # if there's an error
    if predicted_labels[i] != test_labels[i]:
        error_cnt += 1
        # if we failed to detect anomaly
        if predicted_labels[i] == 1:
            false_neg += 1
        # detected anomaly, but it normal
        else:
            false_pos += 1
print ("Total Errors", error_cnt)
print ("False Positives ", false_pos/ float(len(predicted_labels)) * 100, "%")
print ("False Negatives ", false_neg/ float(len(predicted_labels)) * 100, "%")


shortened_predicted = []
shortened_test_label = []
for i in range(len(predicted_labels)):
    if predicted_labels[i] == 1:
        shortened_predicted.append(1)
    else:
        shortened_predicted.append(2)
    if test_labels[i] == 1:
        shortened_test_label.append(1)
    else:
        shortened_test_label.append(2)

short_class_names = ["Normal", "Anomalous"]
```

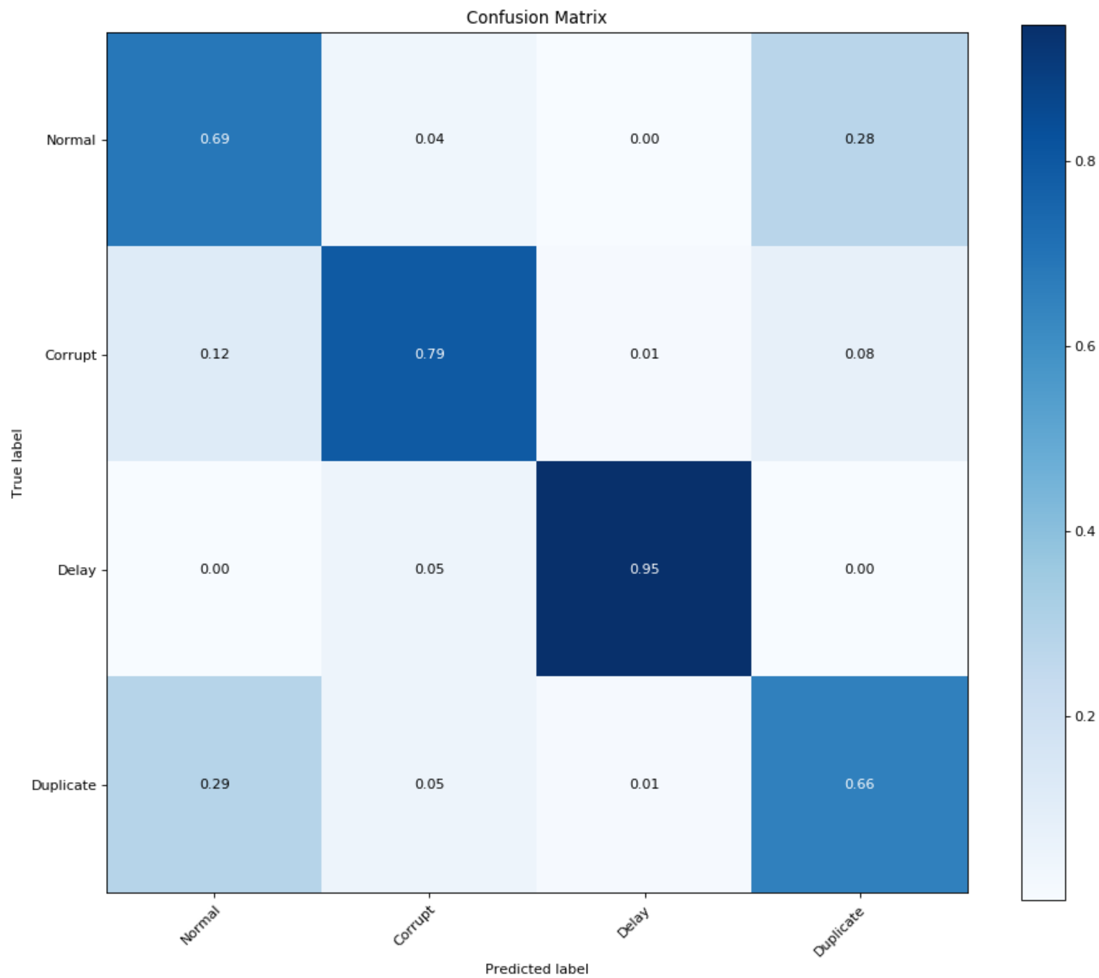



```
Total Error Count :  2343
Normal Class Error Rate :  12.99853491779261 %
Total Error Rate :  19.07048673286668 %
Total Errors 2343
False Positives  8.513755494058278 %
False Negatives  10.5567312388084 %
```

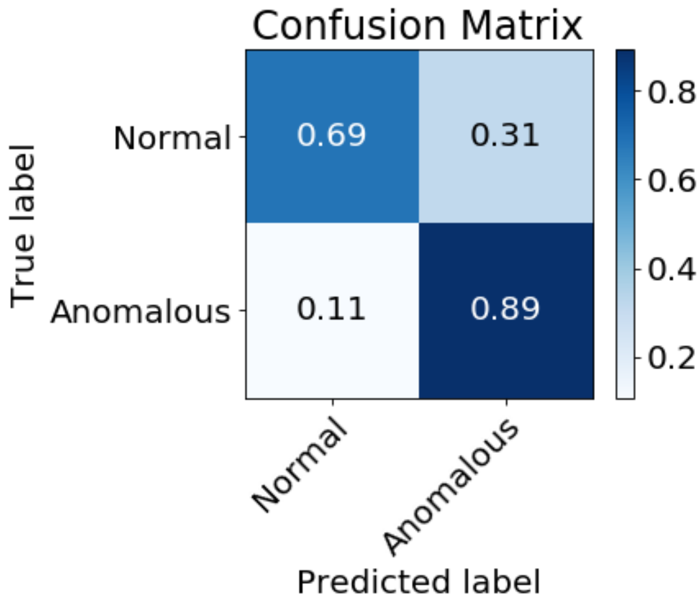


In [28]:
```
from sklearn.model_selection import train_test_split

train_data, test_data, train_labels, test_labels = train_test_split(all_data,
                                                                    anom_type_d
ata_labels, test_size=0.45, random_state=2)
```

In [29]:
```
clf = svm.SVC(kernel='linear', gamma ='auto', max_iter=1000000000, class_weigh
t='balanced')
clf.fit(train_data, train_labels)
predicted_labels = clf.predict(test_data)

from sklearn.metrics import confusion_matrix

class_names = ["Normal", "Corrupt", "Delay", "Duplicate"]

plot_confusion_matrix(test_labels, predicted_labels, normalize=True,classes=cla
ss_names, title='Confusion Matrix')
plt.show()

error_cnt = 0
total_error_cnt = 0
same_class_error = 0
for i in range(len(predicted_labels)):
    if predicted_labels[i] != test_labels[i]:
        total_error_cnt += 1
        # if its normal data
        if predicted_labels[i] == 1 or test_labels[i] == 1:
            error_cnt += 1


print ("Total Error Count : ", total_error_cnt)
print ("Normal Class Error Rate : ", float(error_cnt)/float(len(predicted_label
s)) * 100, "%")
print ("Total Error Rate : ", float(total_error_cnt)/float(len(predicted_label
s)) * 100, "%")


false_pos = 0
false_neg = 0
error_cnt = 0
for i in range(len(predicted_labels)):
    # if there's an error
    if predicted_labels[i] != test_labels[i]:
        error_cnt += 1
        # if we failed to detect anomaly
        if predicted_labels[i] == 1:
            false_neg += 1
        # detected anomaly, but it normal
        else:
            false_pos += 1
print ("Total Errors", error_cnt)
print ("False Positives ", false_pos/ float(len(predicted_labels)) * 100, "%")
print ("False Negatives ", false_neg/ float(len(predicted_labels)) * 100, "%")


shortened_predicted = []
shortened_test_label = []
for i in range(len(predicted_labels)):
    if predicted_labels[i] == 1:
        shortened_predicted.append(1)
    else:
        shortened_predicted.append(2)
    if test_labels[i] == 1:
        shortened_test_label.append(1)
    else:
        shortened_test_label.append(2)

short_class_names = ["Normal", "Anomalous"]
```

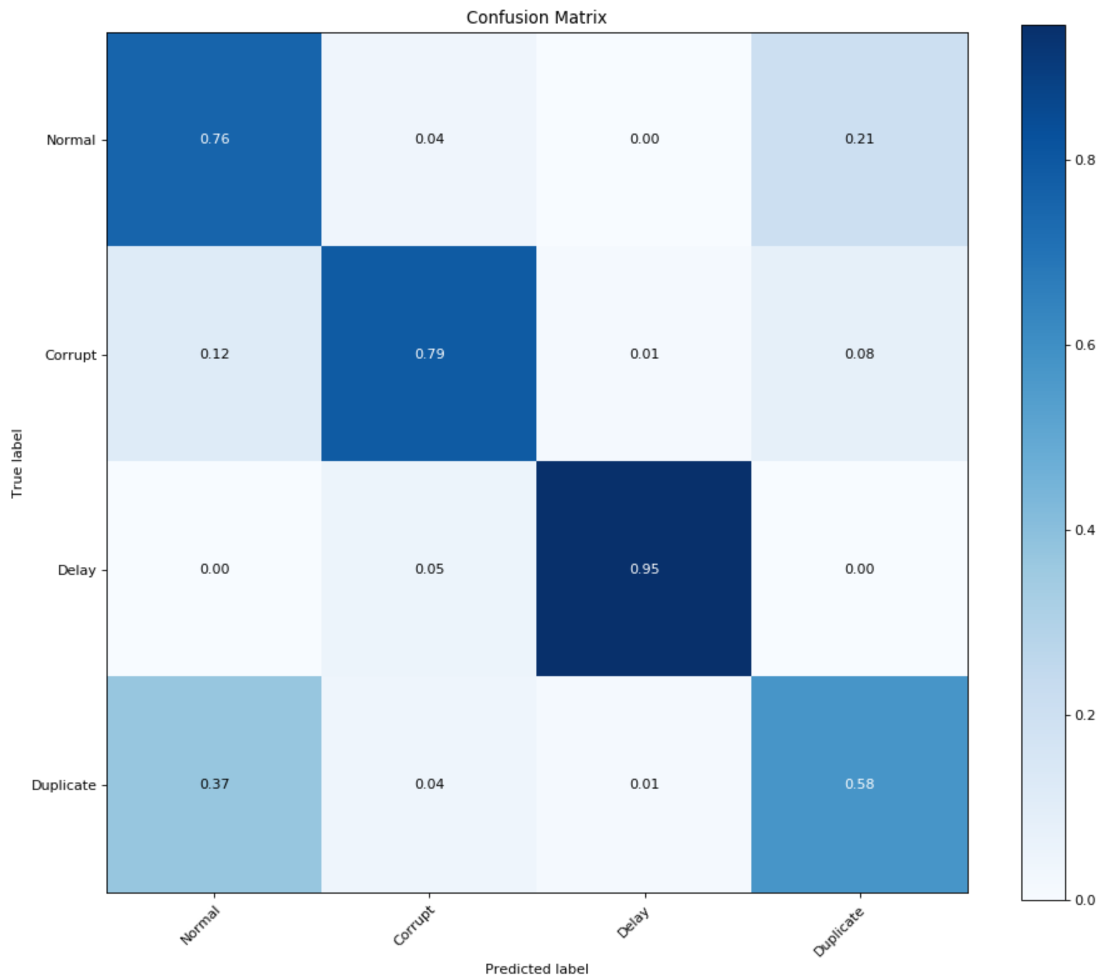



```
Total Error Count :  2309
Normal Class Error Rate :  12.98225622659938 %
Total Error Rate :  18.7937489825818 %
Total Errors 2309
False Positives  9.08350968582126 %
False Negatives  9.710239296760541 %
```

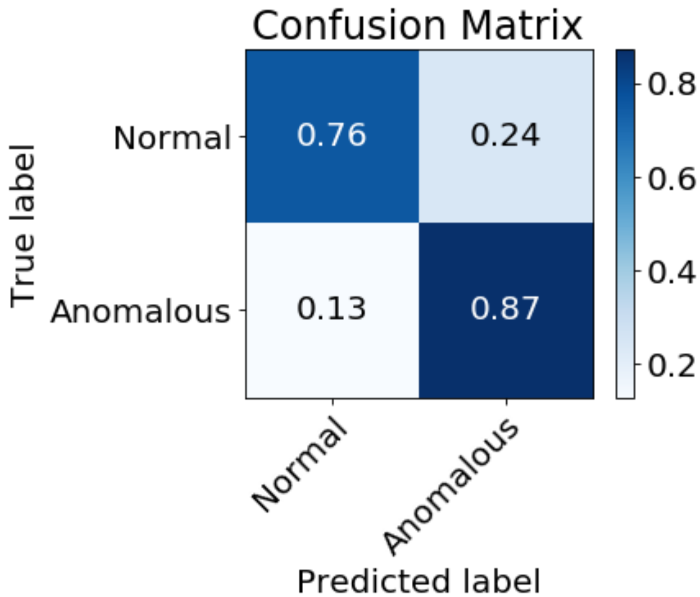


In [30]:
```
from sklearn.model_selection import train_test_split

train_data, test_data, train_labels, test_labels = train_test_split(all_data,
                                                                    anom_type_d
ata_labels, test_size=0.45, random_state=3)
```

In [31]:
```
clf = svm.SVC(kernel='linear', gamma ='auto', max_iter=1000000000, class_weigh
t='balanced')
clf.fit(train_data, train_labels)
predicted_labels = clf.predict(test_data)

from sklearn.metrics import confusion_matrix

class_names = ["Normal", "Corrupt", "Delay", "Duplicate"]

plot_confusion_matrix(test_labels, predicted_labels, normalize=True,classes=cla
ss_names, title='Confusion Matrix')
plt.show()

error_cnt = 0
total_error_cnt = 0
same_class_error = 0
for i in range(len(predicted_labels)):
    if predicted_labels[i] != test_labels[i]:
        total_error_cnt += 1
        # if its normal data
        if predicted_labels[i] == 1 or test_labels[i] == 1:
            error_cnt += 1


print ("Total Error Count : ", total_error_cnt)
print ("Normal Class Error Rate : ", float(error_cnt)/float(len(predicted_label
s)) * 100, "%")
print ("Total Error Rate : ", float(total_error_cnt)/float(len(predicted_label
s)) * 100, "%")


false_pos = 0
false_neg = 0
error_cnt = 0
for i in range(len(predicted_labels)):
    # if there's an error
    if predicted_labels[i] != test_labels[i]:
        error_cnt += 1
        # if we failed to detect anomaly
        if predicted_labels[i] == 1:
            false_neg += 1
        # detected anomaly, but it normal
        else:
            false_pos += 1
print ("Total Errors", error_cnt)
print ("False Positives ", false_pos/ float(len(predicted_labels)) * 100, "%")
print ("False Negatives ", false_neg/ float(len(predicted_labels)) * 100, "%")


shortened_predicted = []
shortened_test_label = []
for i in range(len(predicted_labels)):
    if predicted_labels[i] == 1:
        shortened_predicted.append(1)
    else:
        shortened_predicted.append(2)
    if test_labels[i] == 1:
        shortened_test_label.append(1)
    else:
        shortened_test_label.append(2)

short_class_names = ["Normal", "Anomalous"]
```

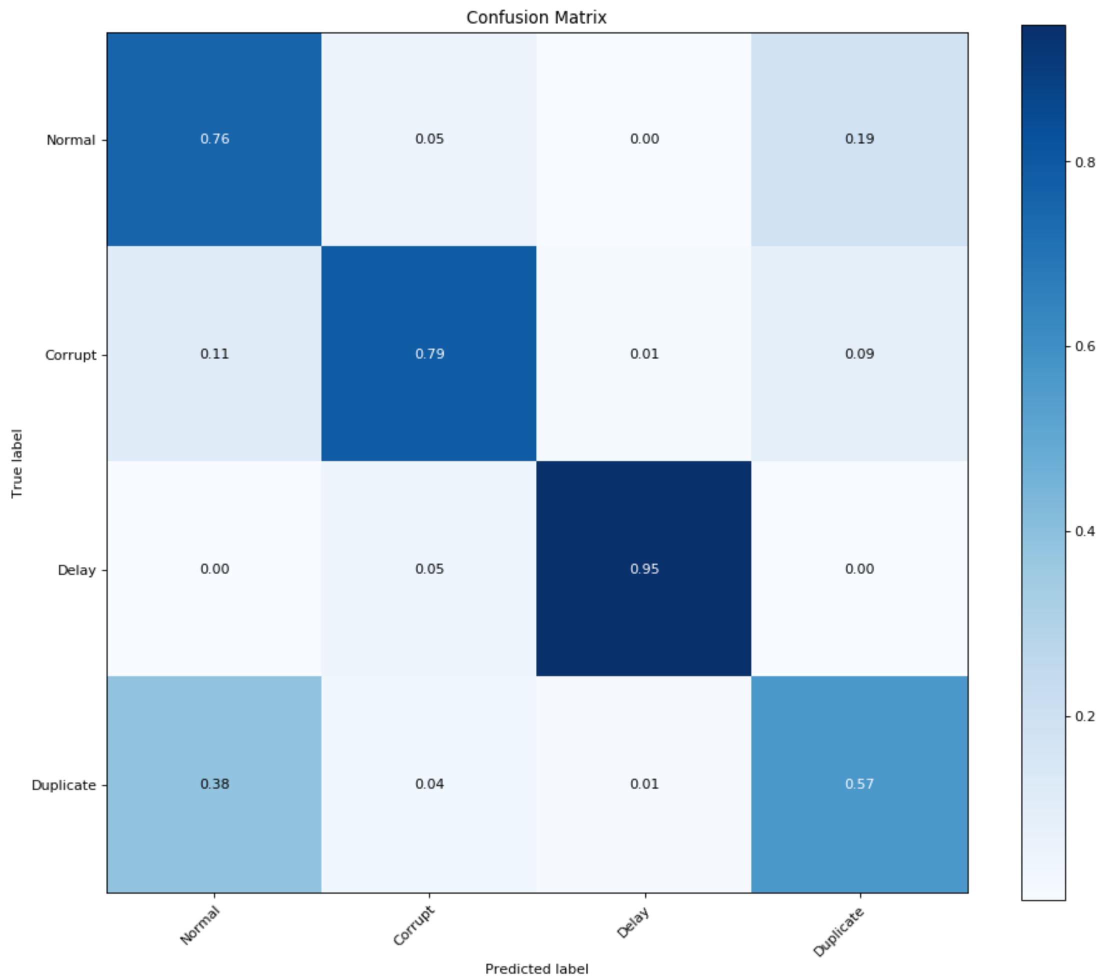



```
Total Error Count :  2458
Normal Class Error Rate :  14.138043301318573 %
Total Error Rate :  20.00651147647729 %
Total Errors 2458
False Positives  8.521894839654893 %
False Negatives  11.4846166368224 %
```

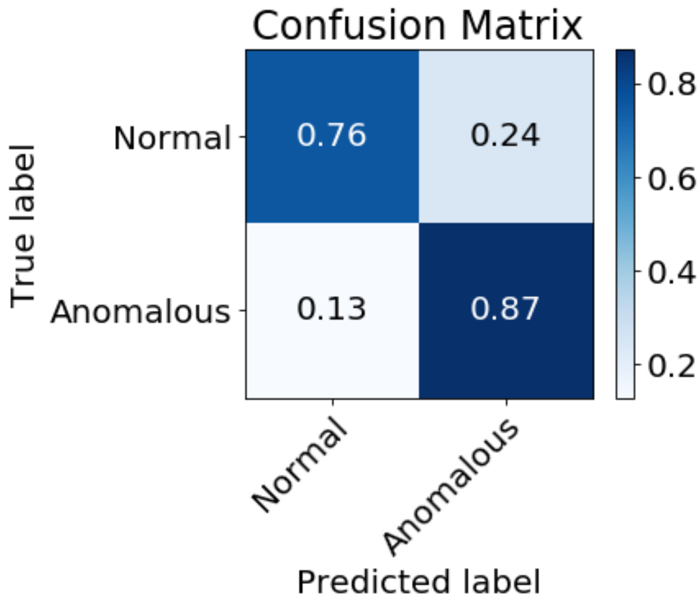


In [32]:
```
from sklearn.model_selection import train_test_split

train_data, test_data, train_labels, test_labels = train_test_split(all_data,
                                                                    anom_type_d
ata_labels, test_size=0.45, random_state=4)
```

In [33]:
```
clf = svm.SVC(kernel='linear', gamma ='auto', max_iter=1000000000, class_weigh
t='balanced')
clf.fit(train_data, train_labels)
predicted_labels = clf.predict(test_data)

from sklearn.metrics import confusion_matrix

class_names = ["Normal", "Corrupt", "Delay", "Duplicate"]

plot_confusion_matrix(test_labels, predicted_labels, normalize=True,classes=cla
ss_names, title='Confusion Matrix')
plt.show()

error_cnt = 0
total_error_cnt = 0
same_class_error = 0
for i in range(len(predicted_labels)):
    if predicted_labels[i] != test_labels[i]:
        total_error_cnt += 1
        # if its normal data
        if predicted_labels[i] == 1 or test_labels[i] == 1:
            error_cnt += 1


print ("Total Error Count : ", total_error_cnt)
print ("Normal Class Error Rate : ", float(error_cnt)/float(len(predicted_label
s)) * 100, "%")
print ("Total Error Rate : ", float(total_error_cnt)/float(len(predicted_label
s)) * 100, "%")


false_pos = 0
false_neg = 0
error_cnt = 0
for i in range(len(predicted_labels)):
    # if there's an error
    if predicted_labels[i] != test_labels[i]:
        error_cnt += 1
        # if we failed to detect anomaly
        if predicted_labels[i] == 1:
            false_neg += 1
        # detected anomaly, but it normal
        else:
            false_pos += 1
print ("Total Errors", error_cnt)
print ("False Positives ", false_pos/ float(len(predicted_labels)) * 100, "%")
print ("False Negatives ", false_neg/ float(len(predicted_labels)) * 100, "%")


shortened_predicted = []
shortened_test_label = []
for i in range(len(predicted_labels)):
    if predicted_labels[i] == 1:
        shortened_predicted.append(1)
    else:
        shortened_predicted.append(2)
    if test_labels[i] == 1:
        shortened_test_label.append(1)
    else:
        shortened_test_label.append(2)

short_class_names = ["Normal", "Anomalous"]
```

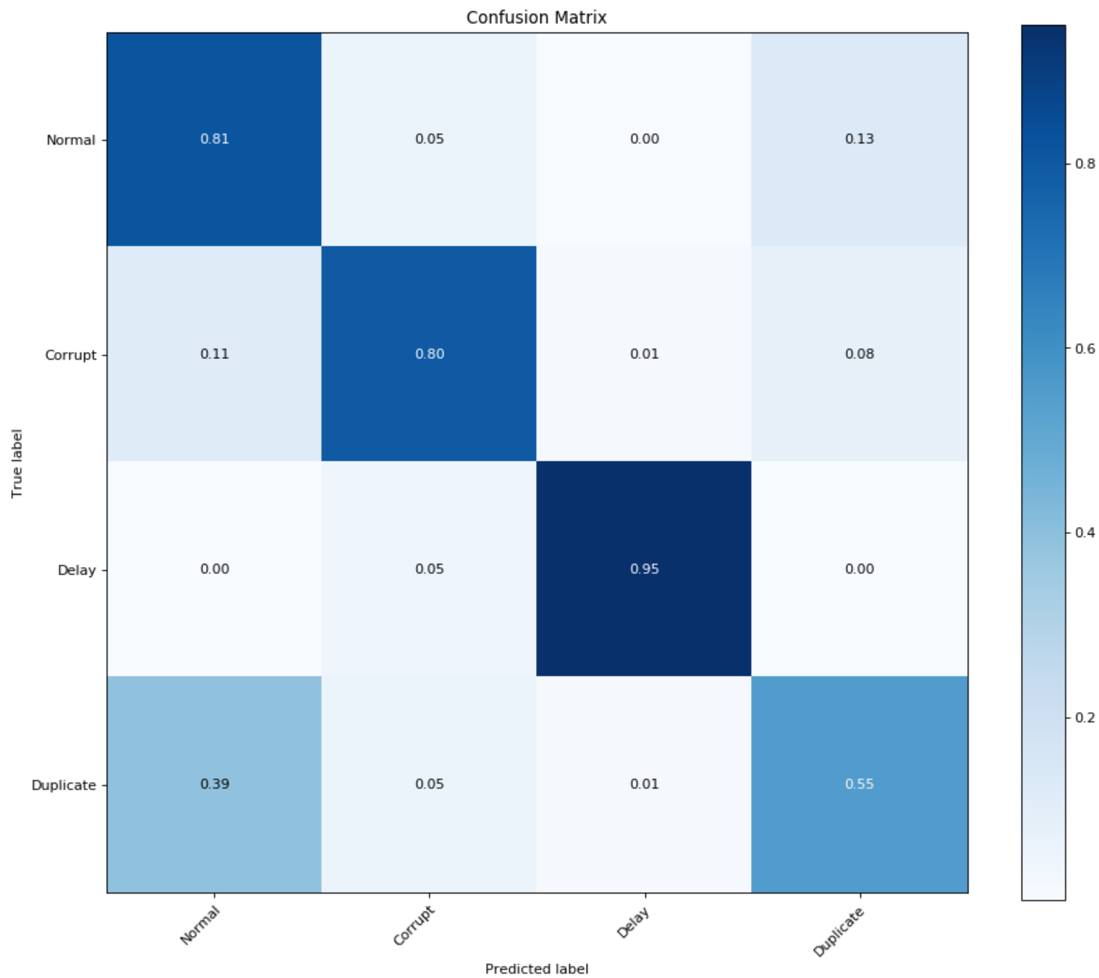



```
Total Error Count :  2472
Normal Class Error Rate :  14.105485918932118 %
Total Error Rate :  20.120462314829886 %
Total Errors 2472
False Positives  8.513755494058278 %
False Negatives  11.60670682077161 %
```

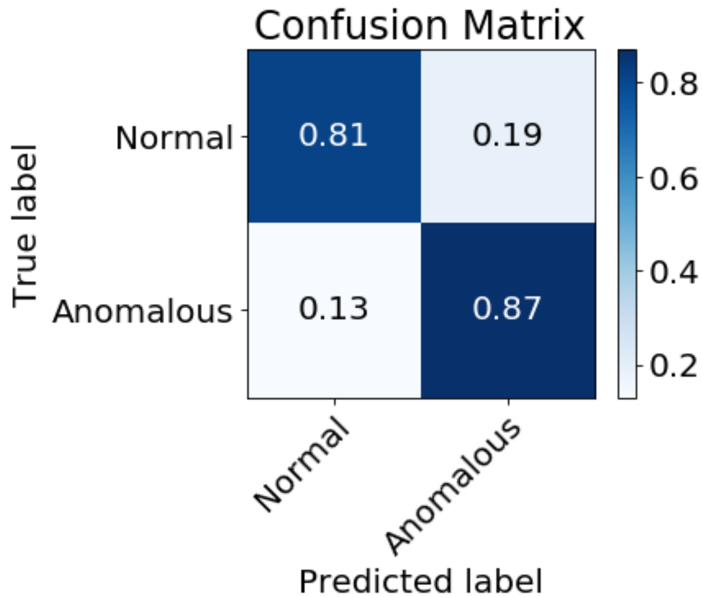


In [34]:
```
from sklearn.model_selection import train_test_split

train_data, test_data, train_labels, test_labels = train_test_split(all_data,
                                                                     anom_type_d
ata_labels, test_size=0.45, random_state=5)
```

In [35]:
```
clf = svm.SVC(kernel='linear', gamma ='auto', max_iter=1000000000, class_weigh
t='balanced')
clf.fit(train_data, train_labels)
predicted_labels = clf.predict(test_data)

from sklearn.metrics import confusion_matrix

class_names = ["Normal", "Corrupt", "Delay", "Duplicate"]

plot_confusion_matrix(test_labels, predicted_labels, normalize=True,classes=cla
ss_names, title='Confusion Matrix')
plt.show()

error_cnt = 0
total_error_cnt = 0
same_class_error = 0
for i in range(len(predicted_labels)):
    if predicted_labels[i] != test_labels[i]:
        total_error_cnt += 1
        # if its normal data
        if predicted_labels[i] == 1 or test_labels[i] == 1:
            error_cnt += 1


print ("Total Error Count : ", total_error_cnt)
print ("Normal Class Error Rate : ", float(error_cnt)/float(len(predicted_label
s)) * 100, "%")
print ("Total Error Rate : ", float(total_error_cnt)/float(len(predicted_label
s)) * 100, "%")


false_pos = 0
false_neg = 0
error_cnt = 0
for i in range(len(predicted_labels)):
    # if there's an error
    if predicted_labels[i] != test_labels[i]:
        error_cnt += 1
        # if we failed to detect anomaly
        if predicted_labels[i] == 1:
            false_neg += 1
        # detected anomaly, but it normal
        else:
            false_pos += 1
print ("Total Errors", error_cnt)
print ("False Positives ", false_pos/ float(len(predicted_labels)) * 100, "%")
print ("False Negatives ", false_neg/ float(len(predicted_labels)) * 100, "%")


shortened_predicted = []
shortened_test_label = []
for i in range(len(predicted_labels)):
    if predicted_labels[i] == 1:
        shortened_predicted.append(1)
    else:
        shortened_predicted.append(2)
    if test_labels[i] == 1:
        shortened_test_label.append(1)
    else:
        shortened_test_label.append(2)

short_class_names = ["Normal", "Anomalous"]
```

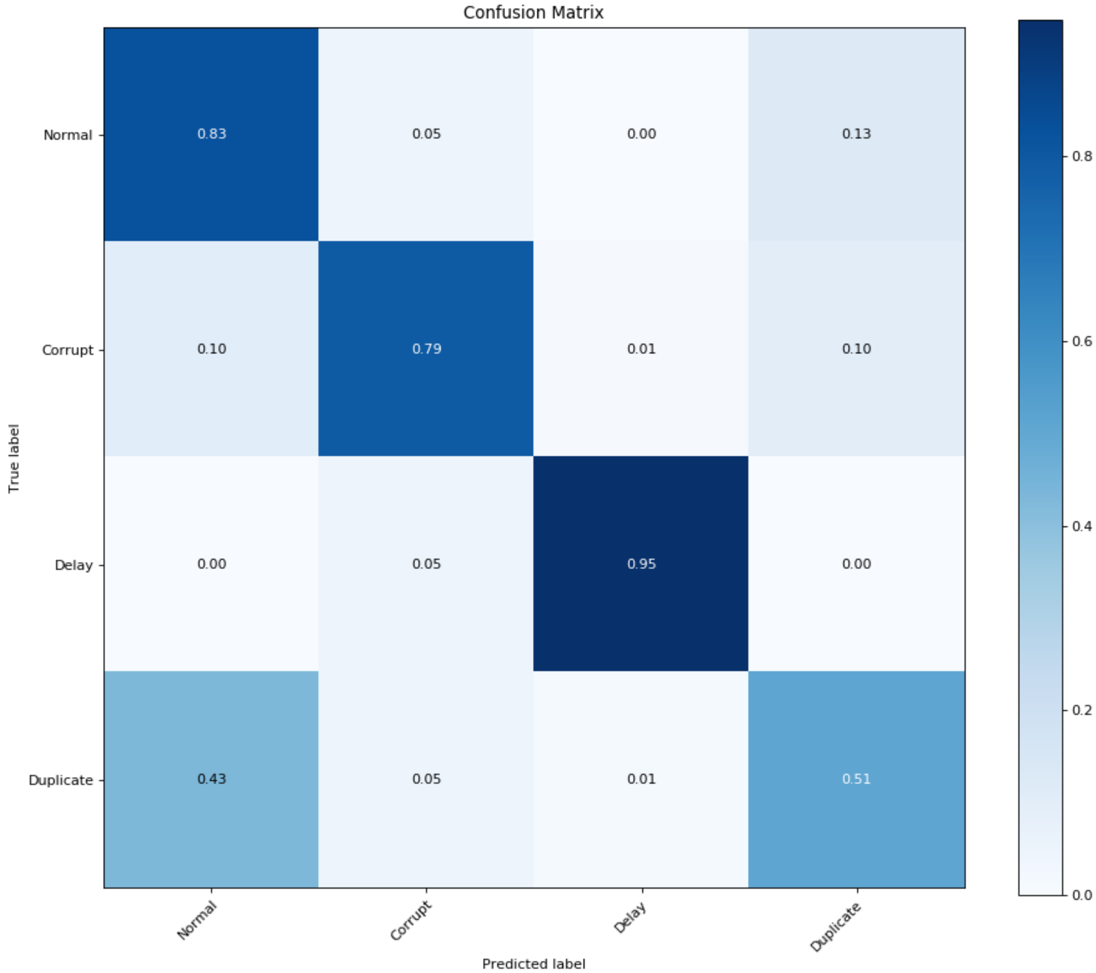



```
Total Error Count :  2400
Normal Class Error Rate :  13.633403874328504 %
Total Error Rate :  19.534429431873676 %
Total Errors 2400
False Positives  7.895165228715611 %
False Negatives  11.639264203158065 %
```

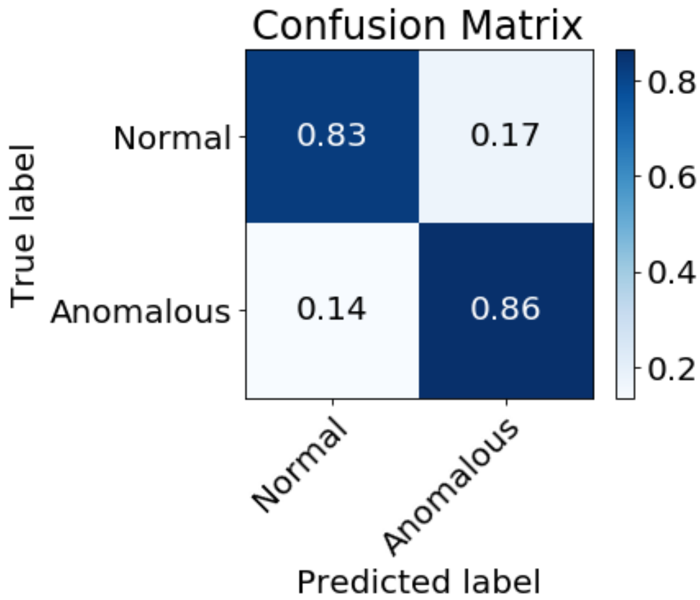


In [36]:
```
from sklearn.model_selection import train_test_split

train_data, test_data, train_labels, test_labels = train_test_split(all_data,
                                                                     anom_type_d
ata_labels, test_size=0.45, random_state=6 )
```

In [37]:
```
clf = svm.SVC(kernel='linear', gamma ='auto', max_iter=1000000000, class_weigh
t='balanced')
clf.fit(train_data, train_labels)
predicted_labels = clf.predict(test_data)

from sklearn.metrics import confusion_matrix

class_names = ["Normal", "Corrupt", "Delay", "Duplicate"]

plot_confusion_matrix(test_labels, predicted_labels, normalize=True,classes=cla
ss_names, title='Confusion Matrix')
plt.show()

error_cnt = 0
total_error_cnt = 0
same_class_error = 0
for i in range(len(predicted_labels)):
    if predicted_labels[i] != test_labels[i]:
        total_error_cnt += 1
        # if its normal data
        if predicted_labels[i] == 1 or test_labels[i] == 1:
            error_cnt += 1


print ("Total Error Count : ", total_error_cnt)
print ("Normal Class Error Rate : ", float(error_cnt)/float(len(predicted_label
s)) * 100, "%")
print ("Total Error Rate : ", float(total_error_cnt)/float(len(predicted_label
s)) * 100, "%")


false_pos = 0
false_neg = 0
error_cnt = 0
for i in range(len(predicted_labels)):
    # if there's an error
    if predicted_labels[i] != test_labels[i]:
        error_cnt += 1
        # if we failed to detect anomaly
        if predicted_labels[i] == 1:
            false_neg += 1
        # detected anomaly, but it normal
        else:
            false_pos += 1
print ("Total Errors", error_cnt)
print ("False Positives ", false_pos/ float(len(predicted_labels)) * 100, "%")
print ("False Negatives ", false_neg/ float(len(predicted_labels)) * 100, "%")


shortened_predicted = []
shortened_test_label = []
for i in range(len(predicted_labels)):
    if predicted_labels[i] == 1:
        shortened_predicted.append(1)
    else:
        shortened_predicted.append(2)
    if test_labels[i] == 1:
        shortened_test_label.append(1)
    else:
        shortened_test_label.append(2)

short_class_names = ["Normal", "Anomalous"]
```




```
Total Error Count :  2523
Normal Class Error Rate :  14.056649845352434 %
Total Error Rate :  20.535568940257203 %
Total Errors 2523
False Positives  8.261435780563243 %
False Negatives  12.27413315969396 %
```


Confusion Matrix
True label
Normal
Anomalous
0.83
0.17
0.14
0.86
0.8
0.6
0.4
0.2
Normal
Anomalous
Predicted label