VoIP-Stage2

March 6, 2020

[1]:
```
import os
import pandas as pd
import numpy as np
import re

import IPython.display as ipd


import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

Load first dataset with scores:

[2]:
```
df_ratings = pd.read_csv('voip-ratings.csv', index_col='Distorted')
# df_ratings.describe()
```

Calculate Tx (outgoing) and Rx (Incoming) bitrates for both caller and callee:

[3]:
```
df_ratings['duration'] = df_ratings['Sample'].str.extract('sample0?(\d+)',␣
↪expand=False).astype(int)
bitrate_cols = []
for role in ['Caller', 'Callee']:
    for xx in ['Rx', 'Tx']:
        df_ratings[xx + 'Bitrate' + role] = df_ratings[xx + 'Bytes' + role] /␣
↪df_ratings['duration'] / 1024.0 * 8
        bitrate_cols.append(xx + 'Bitrate' + role)

# df_ratings[bitrate_cols].describe()
```

Fill empty scores with 1.0, replace incorrect scores with lower/upper bound.

Final score (`ScoreFinal` column) is a weighted sum of different rater scores from first contest stage.

[4]:
```
ScoreColumns =␣
↪['ScoreCombined','ScoreOutput','Score1010','Score1012','Score1002','Score1007','Score997']
for col in ScoreColumns:
    if df_ratings.dtypes[col] != 'float64':
```

```
        df_ratings[col] = df_ratings[col].astype('object').str.replace('^.
↪*ERROR:.*$', '1')
        df_ratings[col] = df_ratings[col].astype('float64')
    df_ratings.loc[df_ratings[col] < 1.0, col] = 1.0
    df_ratings.loc[df_ratings[col] > 5.0, col] = 5.0
    df_ratings[col].fillna(1.0, inplace=True)

df_ratings['ScoreOutput'] = df_ratings[['Score997','ScoreOutput']].max(axis=1)
df_ratings['ScoreFail'] = (df_ratings['ScoreCombined'] <= 1.0) * 1.0;
df_ratings['ScoreFinal'] = df_ratings['ScoreCombined'] * 0.3 +␣
↪df_ratings['ScoreOutput'] * 0.2 + df_ratings['Score1010'] * 0.16 +␣
↪df_ratings['Score1012'] * 0.16  + df_ratings['Score1007'] * 0.16
ScoreColumns.append('ScoreFinal')
df_empty = df_ratings.loc[(df_ratings['ScoreFail'] > 0)]
#df_noempty = df_ratings.loc[(df_ratings['ScoreFail'] <= 0.0)]
df_noempty = df_ratings.copy()
df_noempty.describe()
```

[4]:

	ScorePreprocess	ScoreOutput	ScoreCombined	Score997	Score1010	\
count	7200.000000	7200.000000	7200.000000	7200.000000	7200.000000	
mean	2.996182	3.635982	3.139639	3.438739	2.746697	
std	1.146100	1.210424	1.158049	1.178836	1.136889	
min	1.000000	1.000000	1.000000	1.000000	1.000000	
25%	2.112650	3.024925	2.378225	2.825350	2.000000	
50%	2.933960	4.048040	3.388010	3.849045	2.400000	
75%	4.102830	4.574105	4.019240	4.322715	3.520000	
max	5.000000	4.999480	4.964010	4.982250	5.000000	

	Score1012	Score1002	Score1007	RxBytesCaller	RxPacketsCaller	\
count	7200.000000	7200.000000	7200.000000	7.200000e+03	7200.000000	
mean	3.307957	2.207081	2.516519	7.151906e+04	302.795972	
std	1.466750	1.032740	1.108811	1.132832e+05	451.631819	
min	1.000000	1.000000	1.000000	0.000000e+00	0.000000	
25%	1.596605	1.305000	1.700750	3.952100e+04	163.000000	
50%	3.820470	2.003500	2.272000	6.096800e+04	229.000000	
75%	4.585657	2.859500	3.189250	7.025300e+04	300.000000	
max	5.000000	4.663000	5.000000	1.278654e+06	4749.000000	

	…	RxPacketsCallee	TxBytesCallee	TxPacketsCallee	duration	\
count	…	7200.000000	7.200000e+03	7200.000000	7200.000000	
mean	…	866.757639	7.146419e+04	300.248333	15.460000	
std	…	1217.054110	1.132730e+05	450.821761	1.004261	
min	…	11.000000	1.104000e+03	7.000000	14.000000	
25%	…	303.000000	3.952550e+04	161.000000	15.000000	
50%	…	323.000000	6.102950e+04	228.000000	15.000000	
75%	…	775.250000	7.028650e+04	297.000000	16.000000	
max	…	5343.000000	1.278808e+06	4747.000000	17.000000	

```
       RxBitrateCaller  TxBitrateCaller  RxBitrateCallee  TxBitrateCallee  \
count      7200.000000      7200.000000      7200.000000      7200.000000
mean         36.149681       110.242236       110.450976        36.121559
std          57.058133       169.277267       169.726299        57.052575
min           0.000000         0.000000         0.490349         0.507353
25%          20.013802        38.475138        38.443750        20.043850
50%          31.216309        43.796094        43.787337        31.259077
75%          35.304688        66.594535        67.089193        35.306083
max         587.616728       914.194792       914.341667       587.687500

         ScoreFail   ScoreFinal
count  7200.000000  7200.000000
mean      0.100139     3.040476
std       0.300206     1.033708
min       0.000000     0.980000
25%       0.000000     2.307583
50%       0.000000     3.164646
75%       0.000000     3.766059
max       1.000000     4.864239

[8 rows x 23 columns]
```

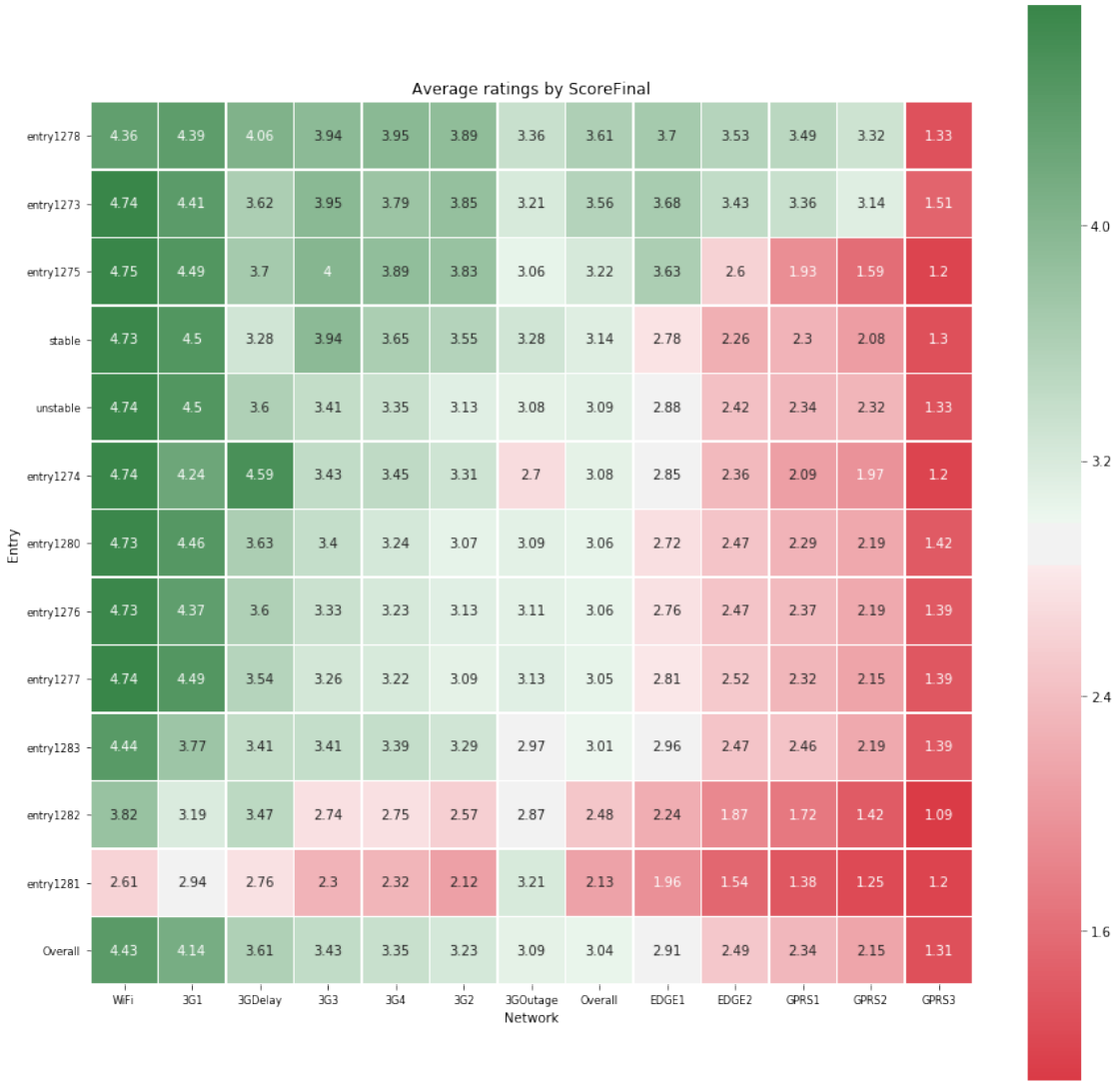
[5]:
```
f, axs = plt.subplots(3, 2, figsize=(25, 35))

def heatmapOnAx(score_col, ax):
    entries_net_ratings = df_noempty.groupby(['Network', 'Entry'])[score_col].
 ↪mean().unstack()
    entries_net_ratings.loc['Overall'] = entries_net_ratings.mean()
    entries_net_ratings = entries_net_ratings.transpose()
    entries_net_ratings = entries_net_ratings.sort_values('Overall',␣
 ↪ascending=False);
    entries_net_ratings.loc['Overall'] = entries_net_ratings.mean()
    networkorder = entries_net_ratings.loc['Overall'].
 ↪sort_values(ascending=False).index.tolist()
    entries_net_ratings = entries_net_ratings[networkorder]
    RateRatingOrder = networkorder;

    cmap = sns.diverging_palette(10, 133, as_cmap=True)
    ax = sns.heatmap(entries_net_ratings, cmap=cmap, square=True, annot=True,␣
 ↪fmt='.3g', linewidths=.5, ax=ax)
    bottom, top = ax.get_ylim()
    ax.set_ylim(bottom + 0.5, top - 0.5)
    ax.set_yticklabels(ax.get_yticklabels(), rotation = 0, fontsize = 8)
    ax.set_xticklabels(ax.get_xticklabels(), rotation = 0, fontsize = 8)
    ax.set_title('Average ratings by ' + score_col)
    return networkorder
```

```
for i, score_col in enumerate(ScoreColumns):
    if i < 6:
        ax = axs[i // 2][i % 2]
        heatmapOnAx(score_col, ax)
```

[6]:
```
f, ax = plt.subplots(figsize=(15, 15))
RateRatingOrder = heatmapOnAx('ScoreFinal', ax)
```



You can listen to audio files: choose 2 entries to compare for some network conditions and launch this cell.

First you will need to download audios from https://data-static.usercontent.dev/VoIP-Stage2-Audios.tar.gz (2.7GB) and extract into audios folder.

[7]:
```
if os.path.isdir("./audios"):
    entry1 = 'stable'
    entry2 = 'unstable'
    net = '3G2'
    scoreType = 'Score1007'
    number = 3

    print(scoreType, 'of', entry1, 'vs', entry2, 'over', net)
```

```
    print('==========================')

    df_check_audios = df_ratings[(df_ratings['Network'] == net) &␣
↪((df_ratings['Entry'] == entry1) | (df_ratings['Entry'] == entry2))];
    df_groups = df_check_audios.groupby(['Sample', 'Entry'], as_index=True)
    df_group_keys = df_groups.groups.keys()
    number *= 2
    for k in df_group_keys:
        number = number - 1
        if number <= 0:
            break
        df_group = df_groups.get_group(k)
        df_group = df_group.sort_values(scoreType)
        row = df_group.iloc[len(df_group)//2]
        filename_pcm = row.name
        file_name_wav = re.sub(r"\.pcm$", ".wav", filename_pcm)
        file_path_wav = 'audio5/' + file_name_wav
        print(scoreType, "%0.2f" % row[scoreType], ', ScoreFinal', "%0.2f" %␣
↪row['ScoreFinal'], file_name_wav)
        ipd.display(ipd.Audio(file_path_wav))
```

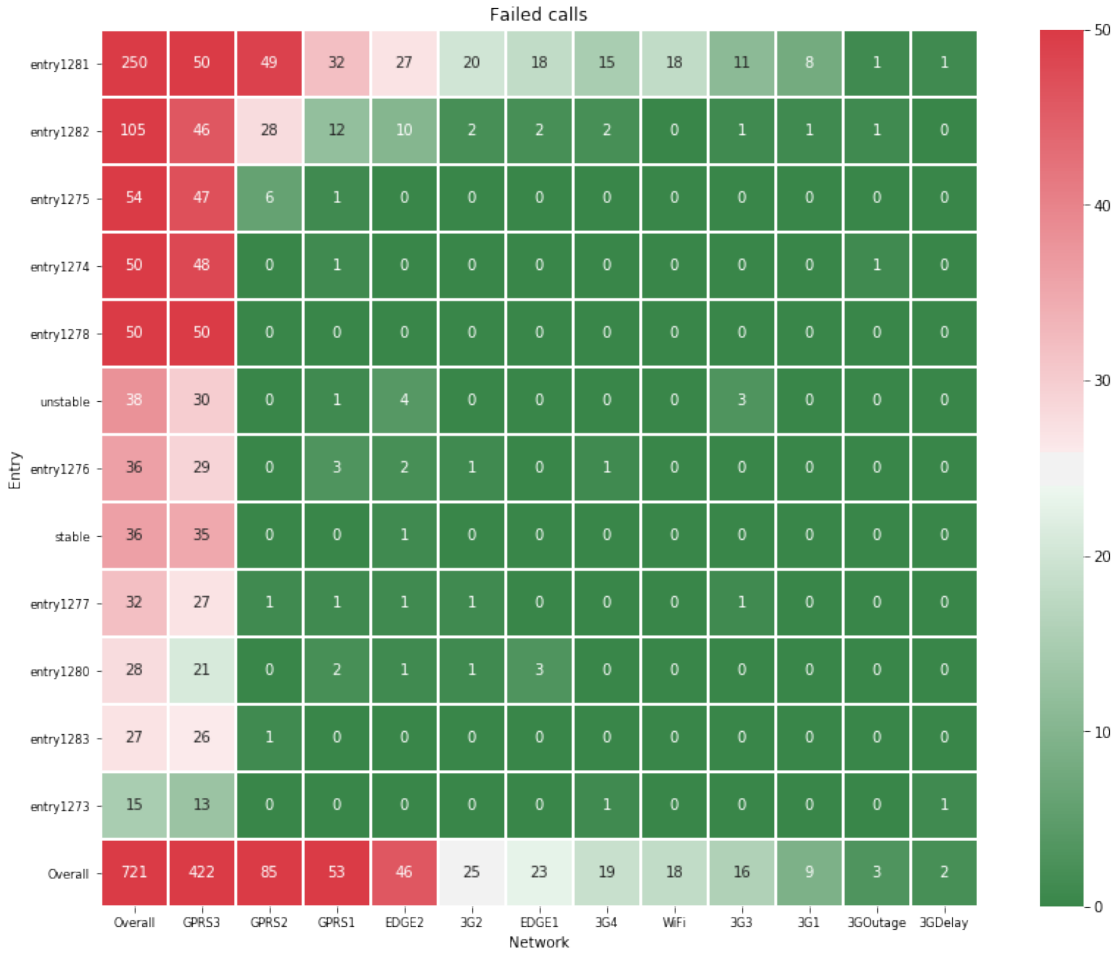
Plot number of failed calls (score <= 1.0):

[8]:
```
entries_net_fails = df_empty.groupby(['Network', 'Entry'])['ScoreFinal'].
↪count().unstack().fillna(0)
entries_net_fails.loc['Overall'] = entries_net_fails.sum()
entries_net_fails = entries_net_fails.transpose()
entries_net_fails = entries_net_fails.sort_values('Overall', ascending=False)
entries_net_fails.loc['Overall'] = entries_net_fails.sum()

networkorder = entries_net_fails.loc['Overall'].sort_values(ascending=False).
↪index.tolist()
entries_net_fails = entries_net_fails[networkorder]


f, ax = plt.subplots(figsize=(15, 11))

cmap = sns.diverging_palette(133, 10, as_cmap=True)
ax = sns.heatmap(entries_net_fails, cmap=cmap, square=True, vmax=50,␣
↪annot=True, fmt='.3g', linewidths=.5)
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
ax.set_yticklabels(ax.get_yticklabels(), rotation = 0, fontsize = 8)
ax.set_xticklabels(ax.get_xticklabels(), rotation = 0, fontsize = 8)
_ = ax.set_title('Failed calls')
```



Plot bitrates:

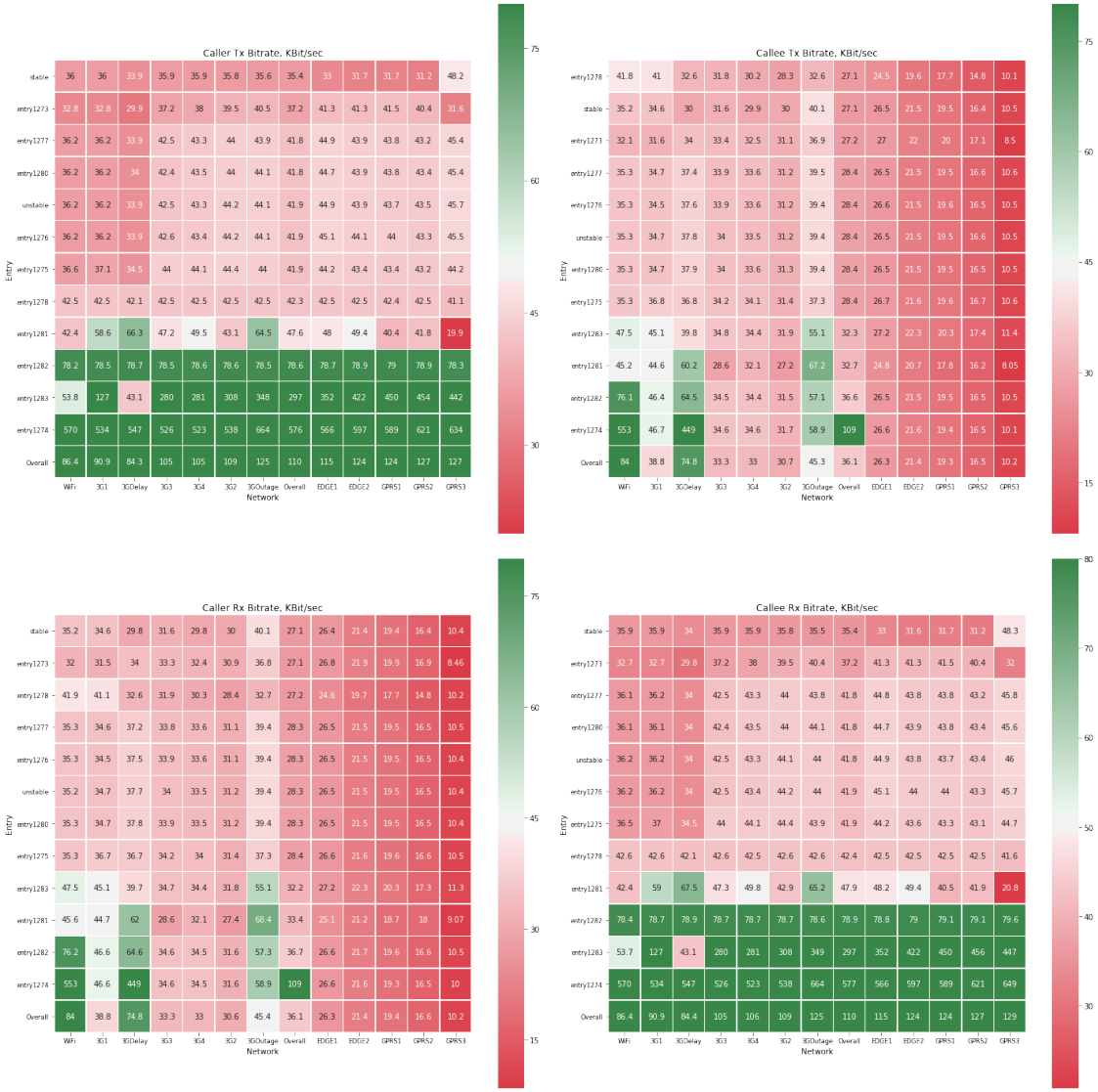
[9]:
```
f, axs = plt.subplots(2, 2, figsize=(20, 20))

plt.tight_layout()

for i, xx in enumerate(['Tx', 'Rx']):
    for j, role in enumerate(['Caller', 'Callee']):
        entries_net_bitrate = df_noempty.groupby(['Network', 'Entry'])[xx +␣
↪'Bitrate' + role].mean().unstack()
        entries_net_bitrate.loc['Overall'] = entries_net_bitrate.mean()
        entries_net_bitrate = entries_net_bitrate.transpose()
        entries_net_bitrate = entries_net_bitrate.sort_values('Overall',␣
↪ascending=True)
        entries_net_bitrate.loc['Overall'] = entries_net_bitrate.mean()
        entries_net_bitrate = entries_net_bitrate[RateRatingOrder]

        cmap = sns.diverging_palette(10, 133, as_cmap=True)
```

```
        ax = axs[i][j]
        ax.set_title(role + ' ' + xx + ' Bitrate, KBit/sec')
        ax = sns.heatmap(entries_net_bitrate, cmap=cmap, square=True,␣
↪annot=True, fmt='.3g', vmax=80, linewidths=.5, ax=ax)
        bottom, top = ax.get_ylim()
        ax.set_ylim(bottom + 0.5, top - 0.5)
        ax.set_yticklabels(ax.get_yticklabels(), rotation = 0, fontsize = 8)
        ax.set_xticklabels(ax.get_xticklabels(), rotation = 0, fontsize = 8)
```



Load separate dataset with timings. This dataset is smaller than the first one and was collected on one host for both caller and callee to minimize time difference.

[10]:
```
df_latency = pd.read_csv('voip-latency.csv')

# Exclude 1281, as it used custom tgvoipcall binary, without timestamps
df_latency = df_latency.loc[(df_latency['ScoreCombined'] > 1.0) &␣
 ↪(df_latency['Entry'] != 'entry1281')]

df_latency['TimeInit'] = df_latency[['TimeInitCaller', 'TimeInitCallee']].
 ↪max(axis=1)
df_latency['TimeConnect'] = df_latency[['TimeFirstWriteCaller',␣
 ↪'TimeFirstReadCaller', 'TimeFirstWriteCallee', 'TimeFirstReadCallee']].
 ↪min(axis=1)
df_latency['TimeEndCaller'] = df_latency[['TimeLastWriteCaller',␣
 ↪'TimeLastReadCaller']].max(axis=1)
df_latency['TimeEndCallee'] = df_latency[['TimeLastWriteCallee',␣
 ↪'TimeLastReadCallee']].max(axis=1)

df_latency['CallerFirstLatency'] = (df_latency[['TimeFirstWriteCaller',␣
 ↪'TimeFirstReadCaller']].max(axis=1) - df_latency['TimeConnect']) / 1000000.0
df_latency['CalleeFirstLatency'] = (df_latency[['TimeFirstWriteCallee',␣
 ↪'TimeFirstReadCallee']].max(axis=1) - df_latency['TimeConnect']) / 1000000.0

df_latency['CallerDuration'] = (df_latency['TimeEndCaller'] -␣
 ↪df_latency['TimeConnect']) / 1000000.0
df_latency['CalleeDuration'] = (df_latency['TimeEndCallee'] -␣
 ↪df_latency['TimeConnect']) / 1000000.0

df_latency['ConnectDuration'] = (df_latency['TimeConnect'] -␣
 ↪df_latency['TimeInit']) / 1000000.0

LatencyColumns = ['CallerFirstLatency', 'CalleeFirstLatency', 'CallerDuration',␣
 ↪'CalleeDuration', 'ConnectDuration']

df_latency.set_index('Distorted', inplace=True)
df_latency.describe()
```

[10]:

	ScorePreprocess	ScoreOutput	ScoreCombined	Score997	Score1012	\
count	1351.000000	1351.000000	1351.000000	1351.000000	1351.000000	
mean	2.884957	3.863857	3.388862	3.831230	3.823127	
std	1.120625	0.956091	0.900001	0.842580	1.269646	
min	1.000000	1.146790	1.024950	1.000000	0.000000	
25%	2.199395	3.457115	2.923240	3.524225	3.558535	
50%	2.921790	4.117200	3.472090	4.052080	4.314260	
75%	4.049590	4.599645	3.997000	4.391705	4.687305	
max	4.637970	4.997970	4.912890	4.900830	5.000000	

	Score1002	Score1007	RxBytesCaller	RxPacketsCaller	\

```
count  1351.000000  1338.000000   1.351000e+03      1351.000000
mean      2.407832     1.765132   7.401819e+04       292.572909
std       1.002918     1.002059   1.167622e+05       457.366004
min       1.000000     0.045000   1.822900e+04        68.000000
25%       1.575000     0.980250   4.273850e+04       165.000000
50%       2.344000     1.577000   6.154200e+04       226.000000
75%       2.975000     2.294750   7.118100e+04       285.000000
max       4.633000     4.144000   1.251304e+06      4700.000000

       TxBytesCaller  …  TimeLastWriteCallee      TimeInit   TimeConnect  \
count   1.351000e+03  …         1.350000e+03  1.350000e+03  1.350000e+03
mean    2.093092e+05  …         1.582859e+15  1.582859e+15  1.582859e+15
std     3.226343e+05  …         2.980598e+10  2.980615e+10  2.980615e+10
min     5.167700e+04  …         1.582822e+15  1.582822e+15  1.582822e+15
25%     7.534500e+04  …         1.582832e+15  1.582832e+15  1.582832e+15
50%     8.486200e+04  …         1.582841e+15  1.582841e+15  1.582841e+15
75%     1.015380e+05  …         1.582892e+15  1.582892e+15  1.582892e+15
max     1.512536e+06  …         1.582901e+15  1.582901e+15  1.582901e+15

       TimeEndCaller  TimeEndCallee  CallerFirstLatency  CalleeFirstLatency  \
count   1.350000e+03   1.350000e+03         1350.000000         1350.000000
mean    1.582859e+15   1.582859e+15            0.147674            0.162875
std     2.980584e+10   2.980598e+10            0.306458            0.308933
min     1.582822e+15   1.582822e+15            0.001112            0.000054
25%     1.582832e+15   1.582832e+15            0.043118            0.043199
50%     1.582841e+15   1.582841e+15            0.043600            0.043785
75%     1.582892e+15   1.582892e+15            0.047051            0.104317
max     1.582901e+15   1.582901e+15            4.485711            4.443070

       CallerDuration  CalleeDuration  ConnectDuration
count     1350.000000     1350.000000      1350.000000
mean        20.274265       20.233533         0.441675
std          1.799981        1.684549         0.590998
min         11.642889       14.059323         0.002370
25%         19.139092       19.130121         0.004557
50%         21.223842       20.929030         0.500702
75%         21.239197       21.240945         0.503852
max         26.902414       27.357211         6.595737

[8 rows x 34 columns]
```

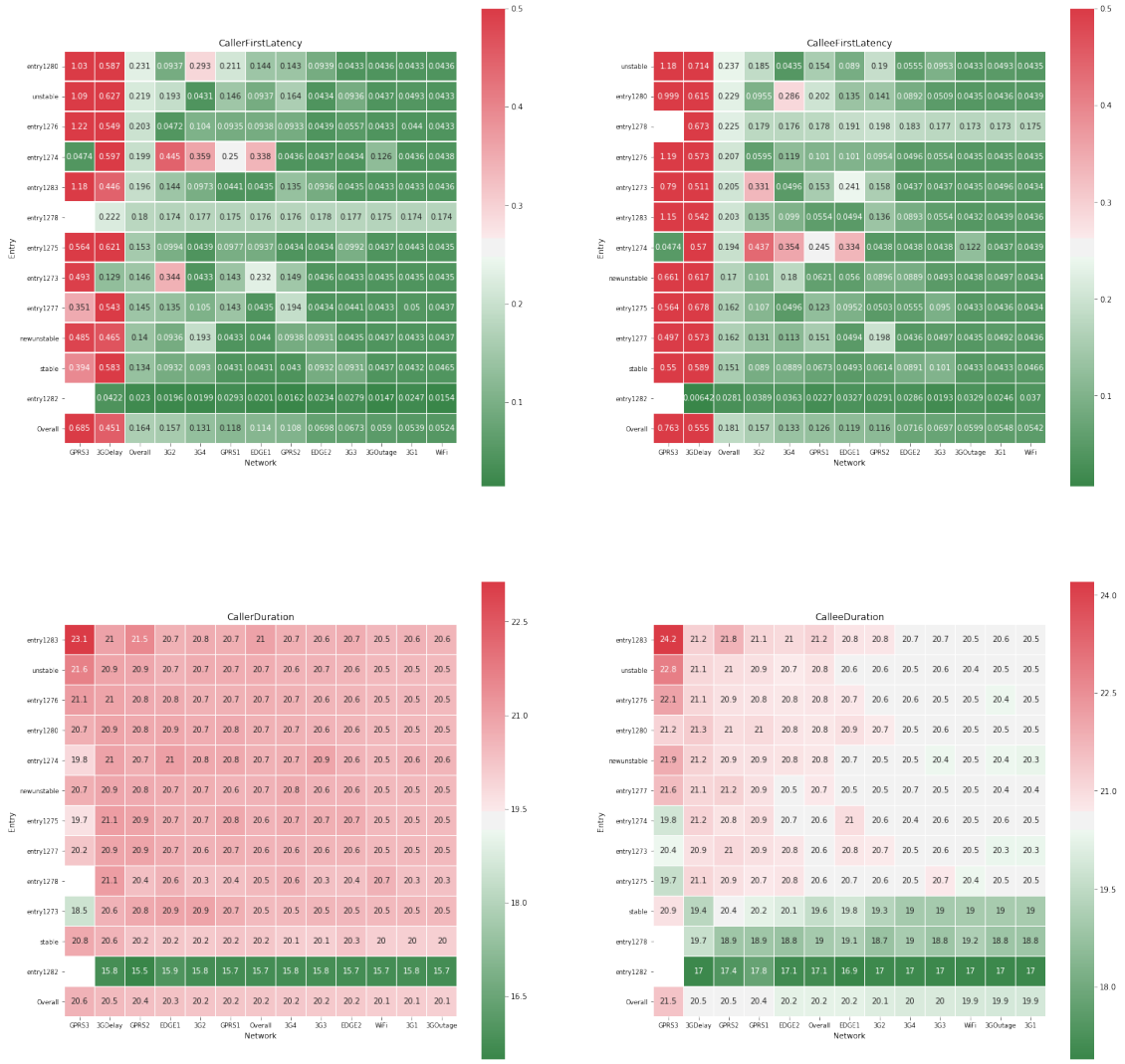
[11]:
```
f, axs = plt.subplots(2, 2, figsize=(25, 25))

def heatmapLatencyOnAx(score_col, ax):
    entries_net_ratings = df_latency.groupby(['Network', 'Entry'])[score_col].
 ↪mean().unstack()
    entries_net_ratings.loc['Overall'] = entries_net_ratings.mean()
```

```
    entries_net_ratings = entries_net_ratings.transpose()
    entries_net_ratings = entries_net_ratings.sort_values('Overall',␣
 ↪ascending=False);
    entries_net_ratings.loc['Overall'] = entries_net_ratings.mean()
    networkorder = entries_net_ratings.loc['Overall'].
 ↪sort_values(ascending=False).index.tolist()
    entries_net_ratings = entries_net_ratings[networkorder]
    RateRatingOrder = networkorder;

#     cmap = sns.diverging_palette(220, 10, as_cmap=True)
    cmap = sns.diverging_palette(133, 10, as_cmap=True)
    vmax = None
    if score_col.endswith('FirstLatency'):
        vmax = 0.5
    elif score_col == 'ConnectDuration':
        vmax = 1
    ax = sns.heatmap(entries_net_ratings, cmap=cmap, square=True, annot=True,␣
 ↪fmt='.3g', linewidths=.5, ax=ax, vmax=vmax)
    bottom, top = ax.get_ylim()
    ax.set_ylim(bottom + 0.5, top - 0.5)
    ax.set_yticklabels(ax.get_yticklabels(), rotation = 0, fontsize = 8)
    ax.set_xticklabels(ax.get_xticklabels(), rotation = 0, fontsize = 8)
    ax.set_title(score_col)
    return networkorder

for i, score_col in enumerate(LatencyColumns):
    if i < 4:
        ax = axs[i // 2][i % 2]
        heatmapLatencyOnAx(score_col, ax)
```



Time taken to establish connection (from tgvoipcall init until first byte read/sent):

[12]:
```
f, ax = plt.subplots(figsize=(15, 15))
c = heatmapLatencyOnAx('ConnectDuration', ax)
```


ConnectDuration
Entry
Network
entry1273 4.63 3.16 1.41 1.42 1.41 1.06 1.01 1.01 0.805 1.01 0.505 0.504 0.506
entry1282 1.42 1.43 0.578 0.333 0.695 0.154 0.533 0.729 0.484 0.129 0.129 0.332
entry1280 1.63 0.663 0.353 0.431 0.503 0.352 0.203 0.352 0.253 0.153 0.203 0.302 0.204
entry1275 0.504 0.657 0.391 0.423 0.551 0.452 0.452 0.402 0.353 0.303 0.402 0.402 0.203
entry1277 1.08 0.807 0.452 0.423 0.303 0.353 0.352 0.108 0.401 0.354 0.202 0.253 0.401
unstable 1.16 0.757 0.253 0.408 0.403 0.253 0.553 0.353 0.153 0.353 0.253 0.203 0.203
newunstable 1.07 0.701 0.303 0.408 0.453 0.402 0.203 0.402 0.353 0.452 0.252 0.153 0.154
entry1274 0.978 0.659 0.225 0.406 0.303 0.391 0.353 0.303 0.252 0.352 0.5 0.252 0.303
stable 1.03 0.767 0.502 0.402 0.452 0.202 0.353 0.353 0.253 0.253 0.202 0.302 0.154
entry1276 0.713 0.609 0.352 0.4 0.453 0.302 0.502 0.402 0.402 0.353 0.303 0.302 0.103
entry1283 0.754 0.761 0.357 0.354 0.302 0.502 0.252 0.103 0.352 0.252 0.202 0.203 0.202
entry1278 0.474 0.054 0.0645 0.0258 0.0184 0.0251 0.0328 0.0368 0.0101 0.0109 0.0111 0.0104
Overall 1.35 0.953 0.506 0.476 0.458 0.415 0.367 0.363 0.362 0.361 0.264 0.251 0.231
GPRS3 3GDelay GPRS2 Overall 3G4 GPRS1 EDGE2 3G2 3G3 EDGE1 3GOutage 3G1 WiFi
1.0
0.8
0.6
0.4
0.2