VoIP-Stage3

April 25, 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_latency = df_ratings.copy()
```

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)
```

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:.*$', '1').str.replace('^[\s\S]*\\n[\s\S]*$', '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]:

	RxBytesCaller	RxPacketsCaller	TxBytesCaller	TxPacketsCaller	\
count	5279.000000	5279.000000	5279.000000	5279.000000	
mean	54683.731578	217.519038	81723.795795	306.791059	
std	29165.118728	79.700170	59056.483744	135.622878	
min	412.000000	5.000000	0.000000	0.000000	
25%	38587.000000	182.000000	49038.000000	263.000000	
50%	52130.000000	221.000000	73902.000000	298.000000	
75%	68305.000000	251.000000	91706.500000	315.000000	
max	316856.000000	612.000000	472794.000000	870.000000	

	RxBytesCallee	RxPacketsCallee	TxBytesCallee	TxPacketsCallee	\
count	5279.000000	5279.000000	5279.000000	5279.000000	
mean	81919.442697	311.933889	54442.853192	213.090358	
std	58955.109797	133.186080	29242.474539	79.850336	
min	412.000000	5.000000	0.000000	0.000000	
25%	49331.000000	264.000000	38494.500000	176.000000	
50%	73842.000000	300.000000	51312.000000	216.000000	
75%	91850.000000	316.000000	68376.000000	250.000000	
max	472628.000000	872.000000	316878.000000	611.000000	

	TimeInitCaller	TimeFirstReadCaller	…	Score1012	Score1002	\
count	5.030000e+03	5.030000e+03	…	5279.000000	5279.000000	
mean	1.586289e+15	1.582504e+15	…	3.739838	2.596874	
std	2.314437e+11	7.739544e+13	…	1.295458	0.871861	
min	1.586112e+15	0.000000e+00	…	1.000000	1.000000	
25%	1.586148e+15	1.586147e+15	…	3.427585	2.074000	
50%	1.586184e+15	1.586184e+15	…	4.205770	2.581000	
75%	1.586220e+15	1.586220e+15	…	4.690205	3.119000	
max	1.586746e+15	1.586746e+15	…	5.000000	4.828000	

	Score1007	duration	RxBitrateCaller	TxBitrateCaller	\

```
count  5279.000000  5279.000000      5279.000000      5279.000000
mean      2.525306    15.487213        27.569693        41.351658
std       0.841529     0.921809        14.583711        30.121341
min       1.000000    14.000000         0.214583         0.000000
25%       1.898667    15.000000        19.493505        25.007292
50%       2.445333    15.000000        26.412500        37.712776
75%       3.016000    16.000000        34.332682        46.863704
max       4.889778    17.000000       155.734375       248.111607

       RxBitrateCallee  TxBitrateCallee    ScoreFail   ScoreFinal
count      5279.000000      5279.000000  5279.000000  5279.000000
mean         41.451192        27.447907     0.087327     3.266336
std          30.070442        14.623508     0.282341     0.990685
min           0.214583         0.000000     0.000000     0.980000
25%          25.004167        19.463450     0.000000     2.864463
50%          37.662946        26.129395     0.000000     3.497836
75%          46.891085        34.364997     0.000000     3.953808
max         247.925223       155.945833     1.000000     4.823238

[8 rows x 33 columns]
```

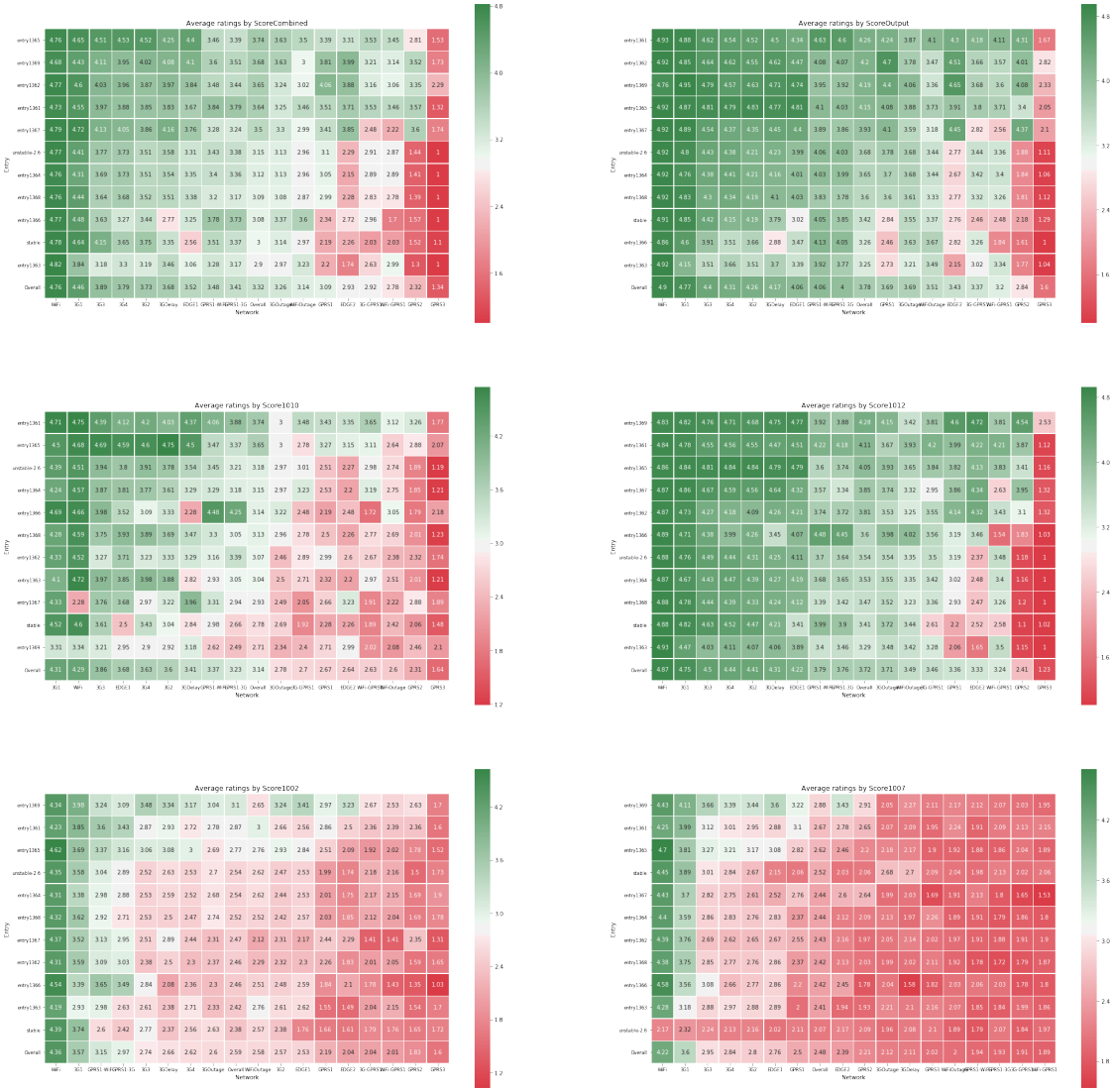
[5]:
```
f, axs = plt.subplots(3, 2, figsize=(35, 35))

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=(20, 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-Stage3-Audios.tar.gz (1.9GB) 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 = 'audios/' + file_name_wav
        print(scoreType, "%0.2f" % row[scoreType], ', ScoreFinal', "%0.2f" %␣
↪row['ScoreFinal'], file_name_wav)
        ipd.display(ipd.Audio(file_path_wav))
```

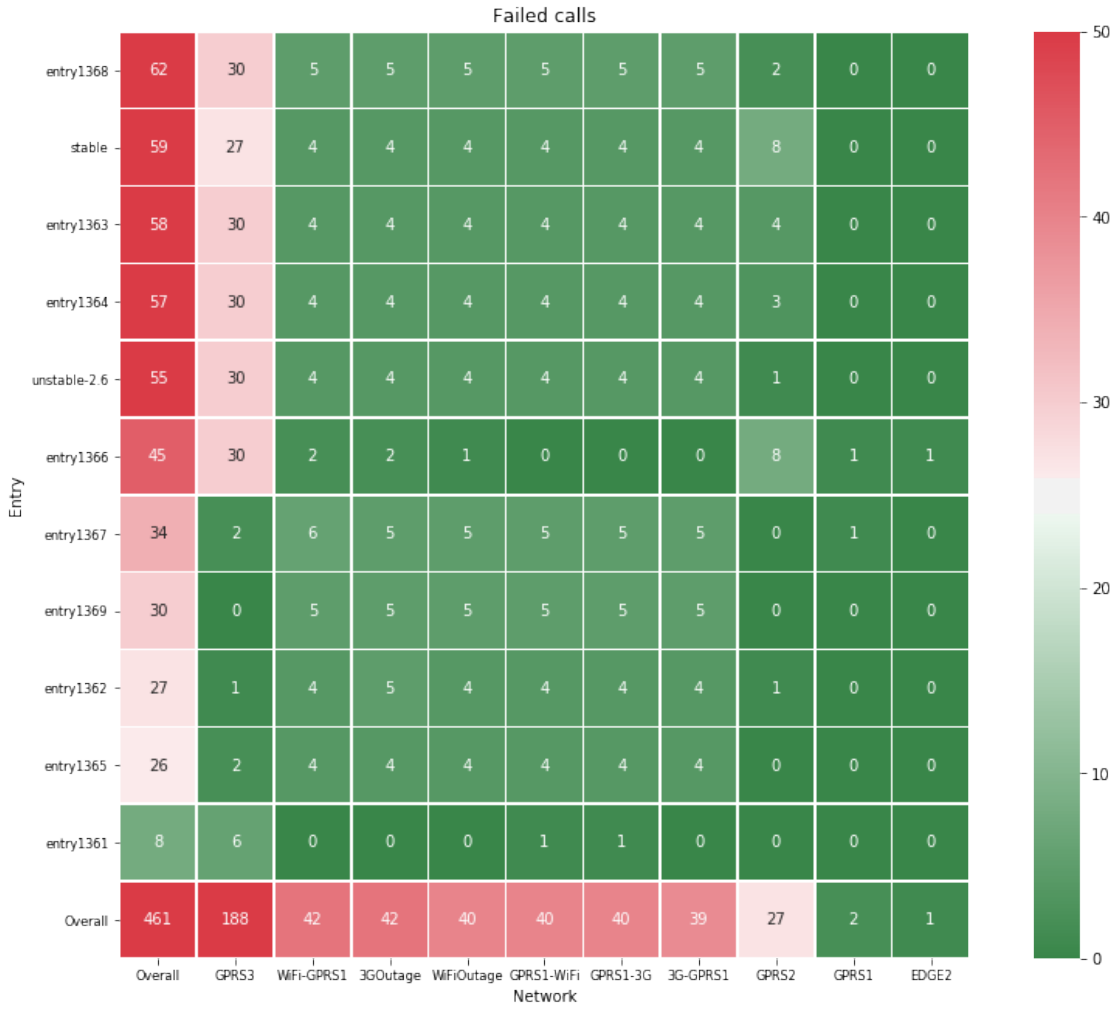
Plot number of failed calls (score <= 1.0):

[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')
```



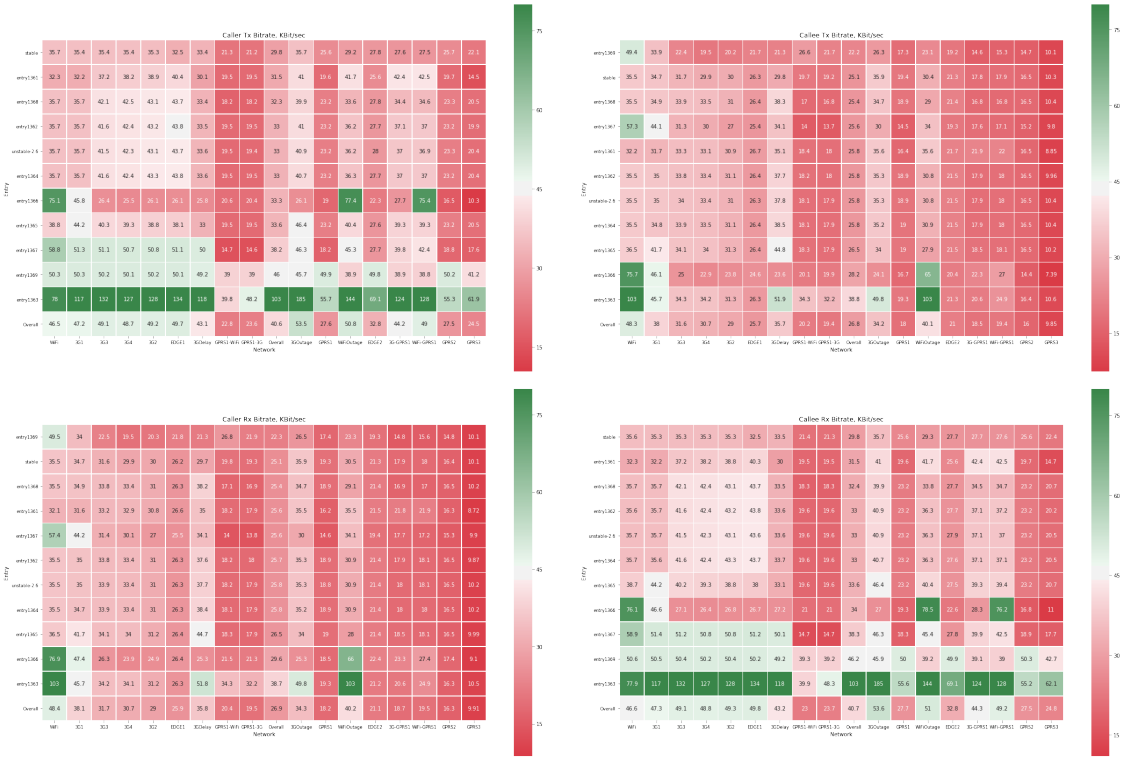
Plot bitrates:

[9]:
```
f, axs = plt.subplots(2, 2, figsize=(30, 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 = df_latency.loc[(df_latency['ScoreCombined'] > 1.0)]

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.describe()
```

[10]:

	RxBytesCaller	RxPacketsCaller	TxBytesCaller	TxPacketsCaller	\
count	4818.000000	4818.000000	4818.000000	4818.000000	
mean	58799.699045	231.378165	87179.965961	321.753010	
std	26867.334413	65.971060	58041.822651	115.408558	
min	14450.000000	91.000000	23781.000000	146.000000	
25%	41923.750000	197.000000	53914.000000	267.000000	
50%	55654.500000	225.000000	77870.000000	307.000000	
75%	69146.000000	256.000000	94001.500000	315.000000	
max	316856.000000	612.000000	472794.000000	867.000000	

	RxBytesCallee	RxPacketsCallee	TxBytesCallee	TxPacketsCallee	\
count	4818.000000	4818.000000	4818.000000	4818.000000	
mean	87318.989207	326.299502	58606.698215	227.333541	
std	57999.454628	113.636922	26877.156861	65.195095	
min	30166.000000	178.000000	13730.000000	85.000000	
25%	54533.750000	267.000000	41681.750000	187.000000	
50%	77803.000000	309.000000	55695.000000	221.000000	
75%	93956.500000	316.000000	69256.000000	255.000000	
max	472628.000000	867.000000	316878.000000	611.000000	

	TimeInitCaller	TimeFirstReadCaller	…	Score1007	TimeInit	\
count	4.817000e+03	4.817000e+03	…	4818.000000	4.817000e+03	
mean	1.586293e+15	1.586293e+15	…	2.623035	1.586293e+15	
std	2.345179e+11	2.345178e+11	…	0.787367	2.345179e+11	
min	1.586112e+15	1.586112e+15	…	1.000000	1.586112e+15	

```
25%      1.586148e+15          1.586148e+15  …     1.973333  1.586148e+15
50%      1.586184e+15          1.586184e+15  …     2.542222  1.586184e+15
75%      1.586221e+15          1.586221e+15  …     3.051556  1.586221e+15
max      1.586746e+15          1.586746e+15  …     4.889778  1.586746e+15

        TimeConnect  TimeEndCaller  TimeEndCallee  CallerFirstLatency  \
count  4.817000e+03   4.817000e+03   4.817000e+03         4817.000000
mean   1.586293e+15   1.586293e+15   1.586293e+15            0.258820
std    2.345179e+11   2.345179e+11   2.345179e+11            0.711353
min    1.586112e+15   1.586112e+15   1.586112e+15            0.000001
25%    1.586148e+15   1.586148e+15   1.586148e+15            0.042179
50%    1.586184e+15   1.586184e+15   1.586184e+15            0.042909
75%    1.586221e+15   1.586221e+15   1.586221e+15            0.130482
max    1.586746e+15   1.586746e+15   1.586746e+15            8.840027

       CalleeFirstLatency  CallerDuration  CalleeDuration  ConnectDuration
count         4817.000000     4817.000000     4817.000000      4817.000000
mean             0.322721       19.886512       19.852049         0.719304
std              0.881402        1.638865        1.541404         0.827479
min              0.000009       10.956017       10.064343        -0.001264
25%              0.043676       18.542128       18.783233         0.491507
50%              0.044888       20.329222       20.094380         0.495311
75%              0.243304       21.231424       21.236965         0.987442
max             10.655206       23.989616       24.592496        16.106398

[8 rows x 35 columns]
```

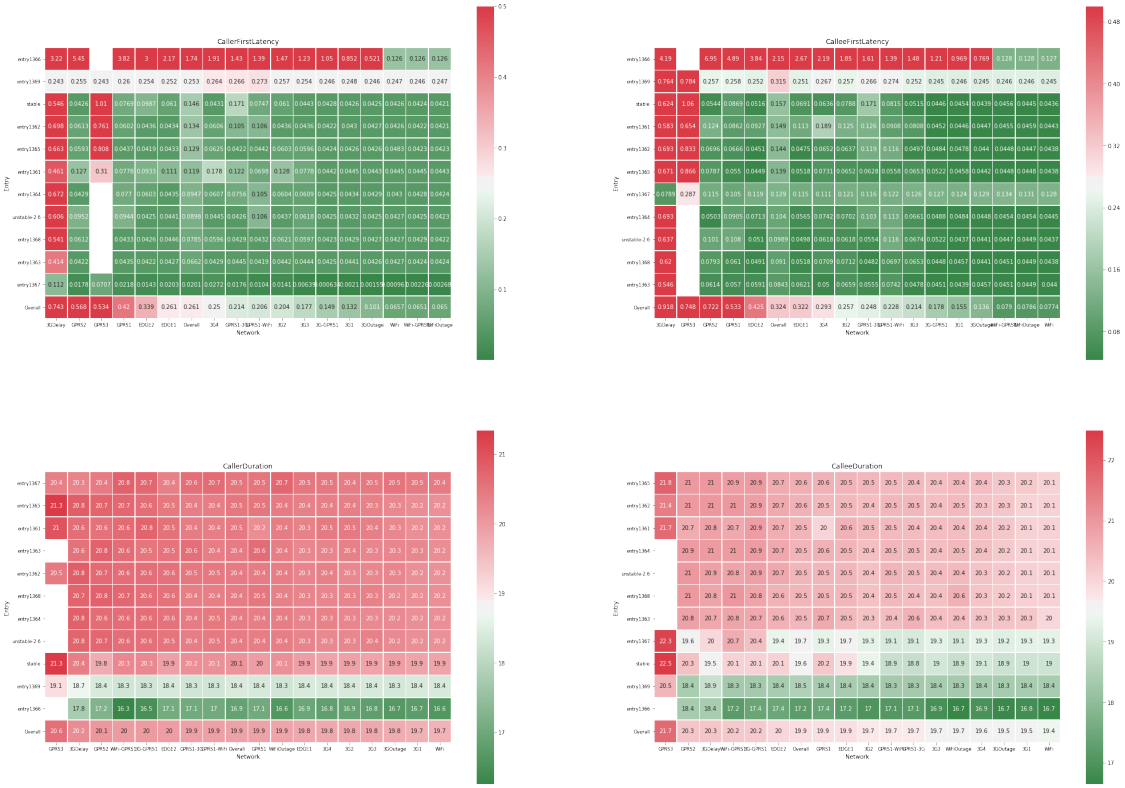
[11]:
```
f, axs = plt.subplots(2, 2, figsize=(35, 25))

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(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=(20, 15))
c = heatmapLatencyOnAx('ConnectDuration', ax)
```


ConnectDuration
Entry
entry1366 5.62 2.07 1.92 1.99 1.94 1.85 1.9 1.53 1.58 1.87 1.56 1.55 1.4 1.16 1.33 1.29 0.935
entry1367 2.46 2.15 1.48 1.5 1.55 1.55 1.35 1.28 1.25 1.38 1.15 1.15 1.02 1.02 1.05 1.02 1.02 1.02
entry1361 4.9 3.03 0.875 0.903 0.62 0.661 1.05 0.847 0.789 0.687 0.788 0.67 0.503 0.504 0.57 0.503 0.503 0.578
entry1365 0.977 0.691 0.61 0.577 0.643 0.557 0.603 0.711 0.627 0.66 0.56 0.643 0.493 0.494 0.493 0.494 0.493 0.524
entry1362 0.923 0.676 0.632 0.577 0.56 0.556 0.578 0.65 0.643 0.511 0.527 0.61 0.493 0.494 0.493 0.494 0.493 0.492
stable 0.685 0.691 0.584 0.56 0.661 0.618 0.567 0.618 0.577 0.56 0.51 0.577 0.495 0.494 0.527 0.494 0.493 0.493
entry1364 0.673 0.571 0.631 0.58 0.623 0.56 0.528 0.697 0.596 0.514 0.58 0.497 0.481 0.512 0.466 0.486 0.528
unstable-2 6 0.604 0.632 0.577 0.56 0.713 0.557 0.587 0.561 0.541 0.576 0.526 0.494 0.493 0.541 0.493 0.494 0.525
entry1368 0.666 0.583 0.644 0.643 0.594 0.553 0.626 0.56 0.493 0.543 0.527 0.493 0.494 0.51 0.493 0.493 0.493
entry1363 0.689 0.0789 0.122 0.0854 0.0591 0.086 0.0328 0.0686 0.0687 0.0197 0.135 0.00247 0.0033 0.00322 0.00255 0.00303 0.00223
entry1369 0.627 0.516 0.0274 0.00872 0.016 0.0165 0.0794 0.0499 0.051 0.0048 0.0164 0.00122 0.000236 0.00219 0.00335 5.64e-05 0.000839 0.00761
Overall 1.76 1.45 0.74 0.729 0.719 0.717 0.713 0.712 0.669 0.644 0.643 0.634 0.549 0.535 0.533 0.526 0.524 0.509
GPRS3 3GDelay GPRS2 GPRS1 3G4 GPRS1-3G Overall GPRS1-WiFi 3G2 EDGE2 EDGE1 3G3 WiFi-GPRS1 WiFi 3G1 WiFiOutage 3GOutage 3G-GPRS1
Network
1.0
0.8
0.6
0.4
0.2