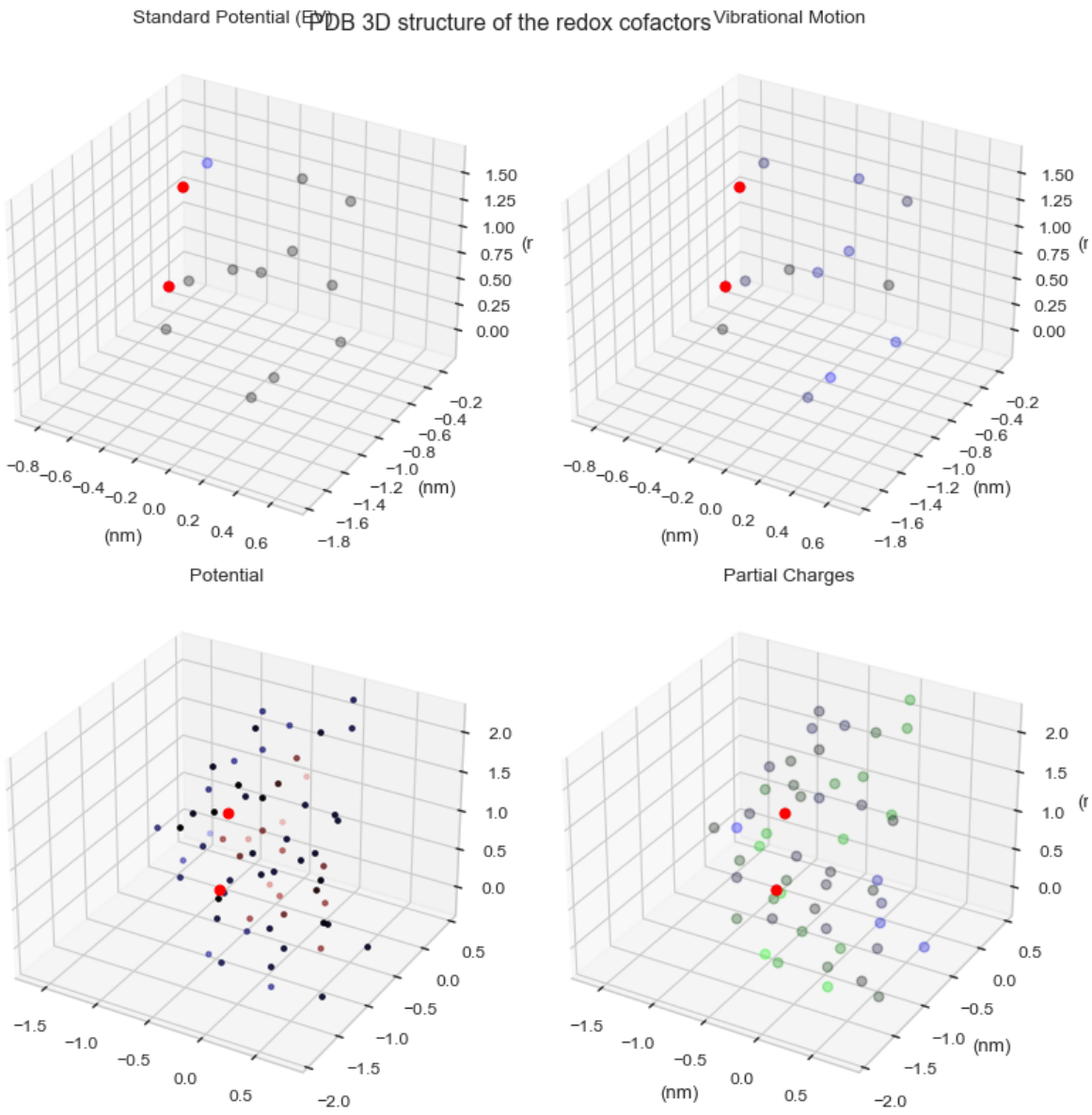
In [1]:

```
#!pip install biopython
#!pip install networkx
#!pip install seaborn
#!pip install griddataformats

import numpy as np
import KMC_Protein
from KMC_Protein import hbar_eV, kbT_eV,electronCharge
from numpy import random
rng= random.default_rng()
import matplotlib.pyplot as plt
import os
import scipy
import numpy as np
import networkx as nx
import seaborn as sns
import KMC_Plotting



sns.set_style("whitegrid")
sns.set_context("paper")

pdb_file = r'./CTPRFixed4.pdb'


#comment out the following line if amino is not desired.
redoxEnergies = {'TRP': .85,
                 'TYR': 1.08, #1.08,
                 #'PHE': 1.15,  #  This is just a guess value from a few papers (Ch
                 'CYS': .85, #1.12,
                 #'HIS': 1.31,
                 #'MET': 1.43,
                 #'BTN': 1.03
                 }  # https://iopscience.iop.org/article/10.1149/2.1471714jes

maxInteraction_radius_nm = 1.6  # nm   this will be used to determine neighbors of

vibrationRadius_nm = .2  # nm this is a global value for the radius of the vibratio

dutton_radius_nm = 1.35 # nm maximum distance between the redox cofactor that allow

reorgE_EV=.2  #standard reorganization energy in eV

injectionEnergy = 5e-6 # eV  energy difference from metal to the injection amino ac
injectionReorg_EV = .16 # eV reorganization energy for the injection amino acid

injectionAminos = [{'residue':'TRP', 'model':0, 'chain':'A', 'index':35} ]

exitAminos  =  [{'residue':'TYR', 'model':0, 'chain':'A', 'index':88} ]

appliedVoltage_V = .2 #V  #assumed to be the voltage at the electrodes

attemptFrequency = 2*np.pi/hbar_eV/np.sqrt( 4*np.pi*reorgE_EV*kbT_eV  )   # attempt
```

```
beta = 1/(kbT_eV)  # 1/kT in 1/eV

print('Loading PDB file...', pdb_file)

#estimate the center of mass for each amino acid
atom_COM,activeAminos = KMC_Protein.LoadAminoCenterOfMass(pdb_file, redoxEnergies,i

KMC_Plotting.PlotPDBProjections(activeAminos,atom_COM,injectionAminos,exitAminos)

G0 = 2*np.power(electronCharge,2)*injectionEnergy/hbar_eV/np.sqrt( np.pi * injectio

injectPrefactor =  2*np.pi/hbar_eV/np.sqrt( 4*np.pi*reorgE_EV*kbT_eV  ) *np.exp(-.3

print(f"{injectPrefactor:.3e} 1/s {1/injectPrefactor*1e9:.3e} ns" )
```

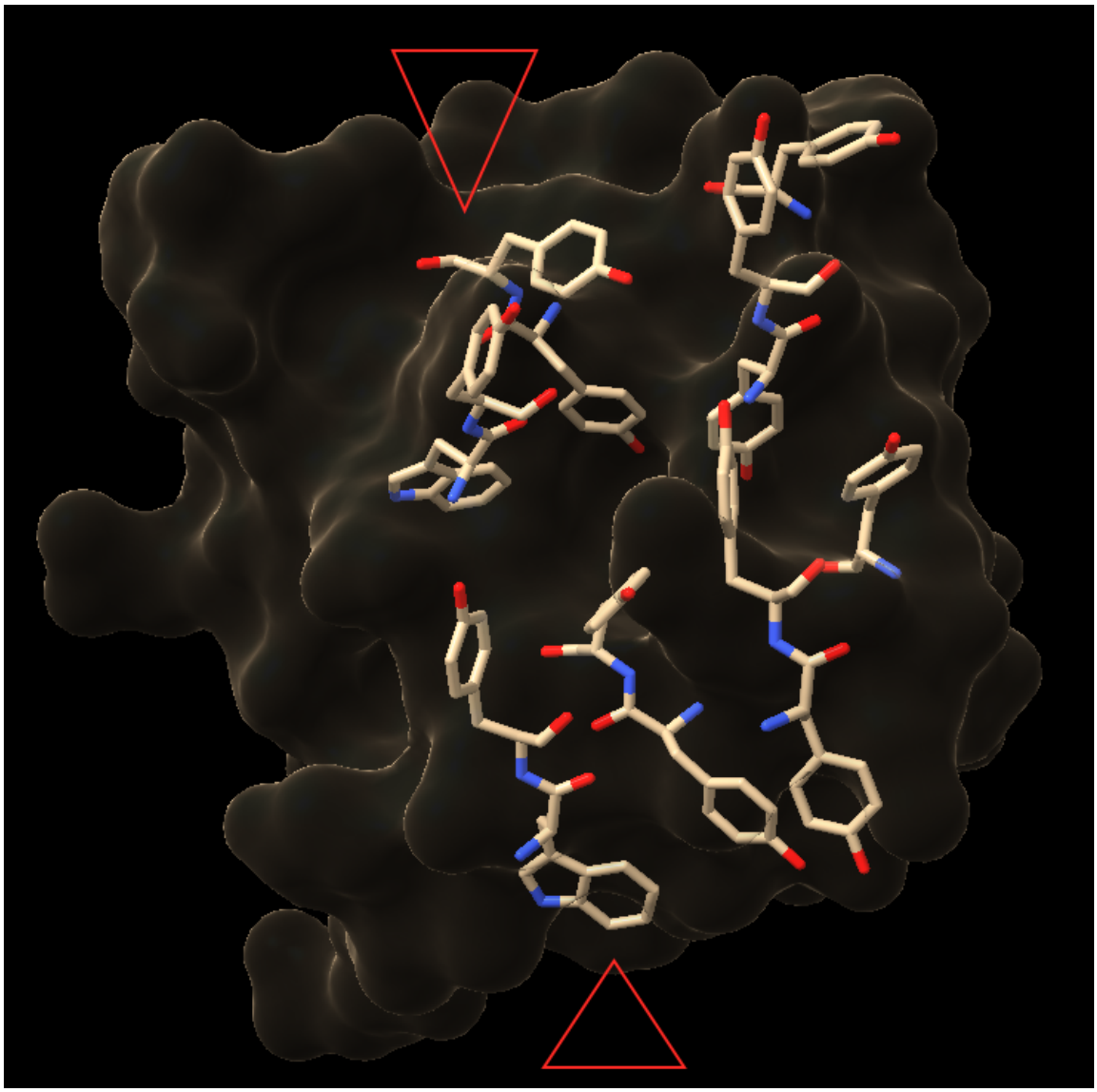
```
Loading PDB file... ./CTPRFixed4.pdb
dx file found
Found Injection Node 0
Found Exit Node 12
A has 68 amino acids with 14 redox cofactors
Find ChargeFile at ./CTPRFixed4.charge
Found charge file: ./CTPRFixed4.charge
Find vibrations at ./CTPRFixed4.std
Found std file: ./CTPRFixed4.std
```



`7.273e+10 1/s 1.375e-02 ns`

CTPR 4

We are showing the center of mass of all the redox active residues in the CTPR molecule to allow comparison with the published values in Long-Range Conductivity in Proteins Mediated by Aromatic Residues



CTPR4

We are looking at the conductivity of this constructed protein, The electron is inserted into the structure at the base of the two streptavidins at the Amino acid location marked by red The above graphs show the various parameters that have been calcuted about this protein structure from other tools.

Standard Potential

The redox factors are taken from literature and assumed to be unaltered by the protein environment.

Potentials

Potentials have been calcuated from Adaptive Poisson-Boltzmann Solver - literature

Vibrational Motion

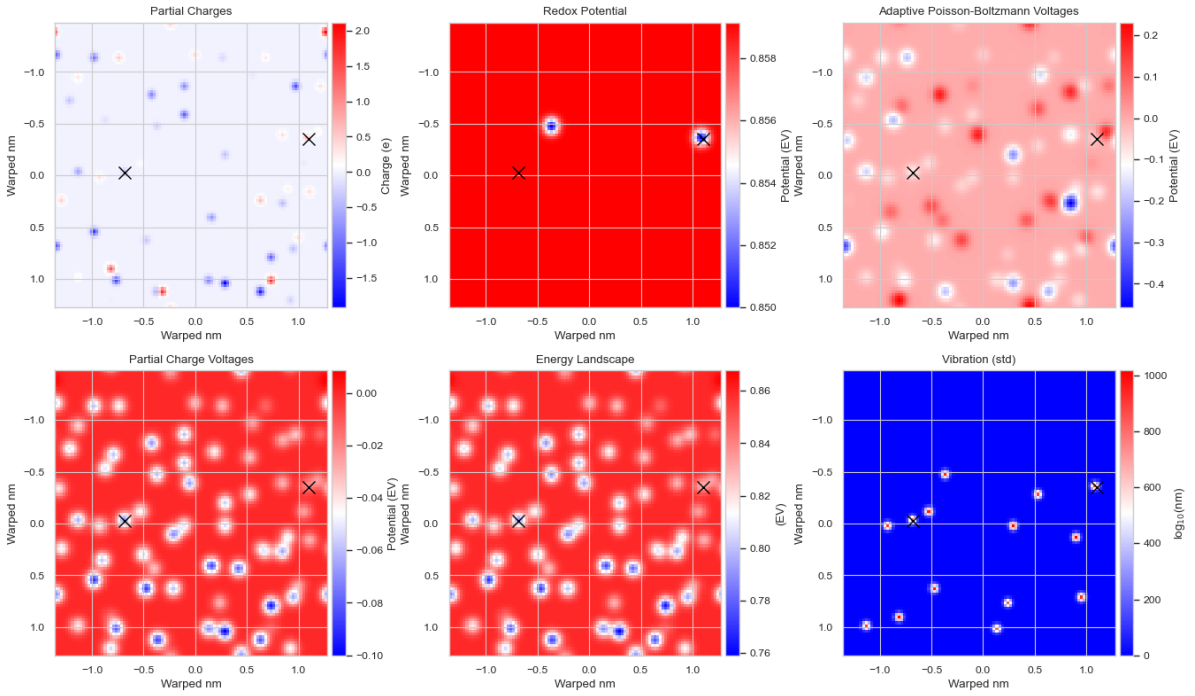
The Vibrational Motion has been determined by running CG simulations in CafeMol - literature using beads on a chain with charges derived from the RESPAC

Partial Charges

Partial charges of the chain as estimated by APBS and the RESPAC method

In [2]:
```
field, fieldZero =KMC_Protein. CalculatePotentials(atom_COM,activeAminos,injectionA
KMC_Plotting.CreateProteinManifold(atom_COM, verbose=True)

KMC_Plotting.PlotPotentialMap(atom_COM )
```

```
Driving force for electron transfer in molecule is 0.01 V/nm
Potentials mapped
Embedding Found
Total pot (eV) 0.008556954891164389 -0.09922570019094072
charges (e) 2.1155 -1.9152
Smoothing Energy Map
d:\PythonProj\KineticMontiCarlo\KMC_Plotting.py:226: RuntimeWarning: divide by zer
o encountered in log10
  vibrateStruct = np.max(np.log10( vibrationPoints))/np.max(vibrateStruct)*vibrate
Struct
```



Manifold graphs

3D structure of the protein complex was reduced to 2D using a manifold esimate to allow easier viewing of the energy landscape. Most of the relationships between the different redox cofactors have been preserved. The inject and removal points have been marked with X allowing visualization of pathways that may form.

Estimation of the driving bias

We consider the self potential of the protein as derived from the CHARMM force field through the APBS program. The induced charges are then added to give the potential from the charged amino acids inside the protein. Finally, the redox potentials of the various amino acids are added to the landscape.

The driving bias from the applied voltage is approximated by using this graph (CTPR supplimental) [https://pubs.acs.org/doi/suppl/10.1021/acsphyschemau.3c00017/suppl_file/pg3c00017_si_001.p

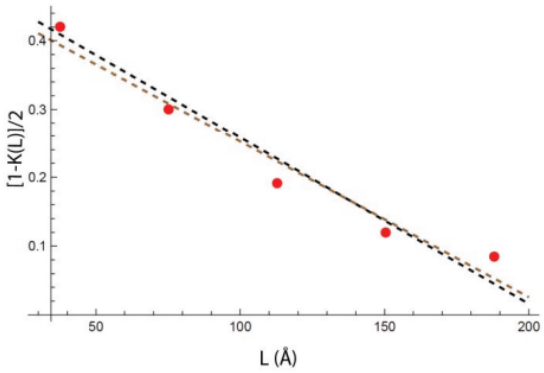
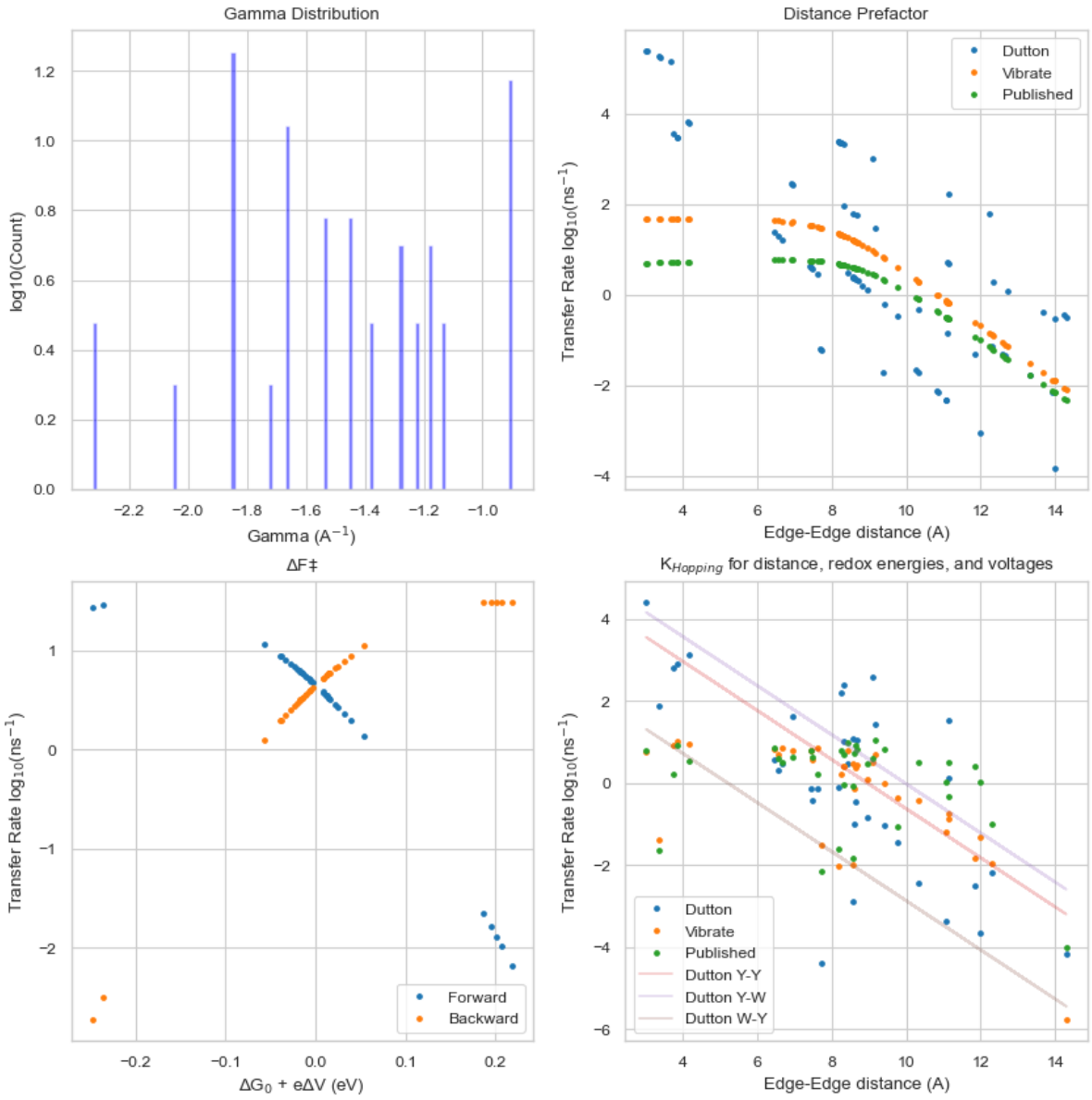


Figure S2: Linear fit to the experimentally-determined bias distribution. K(L) is the fraction of the applied bias that appears across the molecule. The potential drop at the contacts is $\eta(L) = V_b \frac{1-K(L)}{2} \approx V_b(0.5 - 0.02042^{L/\text{Å}})$.

In [3]:
```
##########################################################################################
#                 Determine the transfer rates for each redox cofactor pair
#
##########################################################################################
G_static, G_vibrate, G_min, G_connected, plotValues =KMC_Protein. ConnectGraphs(act
                                                                                att

(gammas, distanceRates, energyRates, transferRates, voltageRates)=plotValues
KMC_Plotting.PlotGraphRates(reorgE_EV,gammas, distanceRates, energyRates, transferR
#del plotValues, gammas, distanceRates, energyRates, transferRates, voltageRates
```

```
min neighbors: 12
max neighbors: 26
mean neighbors: 19.714285714285715
scale metric of graph: 57848.0
local bridges: 0
std neighbors: 3.98978286964827
number of redox cofactors: 14
number of tunnel gaps: 138
```



Rate Calculations

We are assuming that the jumps performed between each redox cofact can be modeled by Marcus theory with each jump modeled by the following equation

$$k_{et} = \frac{2\pi}{\hbar}|H_{AB}|^2 \frac{1}{\sqrt{4\pi\lambda k_B T}} \exp(-\frac{\Delta F^\dagger}{k_B T})$$

where

$\Delta F^\dagger = \frac{(\lambda+\Delta G^\circ+e\Delta V)^2}{4\lambda k_B T}$ as shown in the Gibb's change in the graph above. As we do not have precise calculations of ΔG°, the redox standard potentials are used

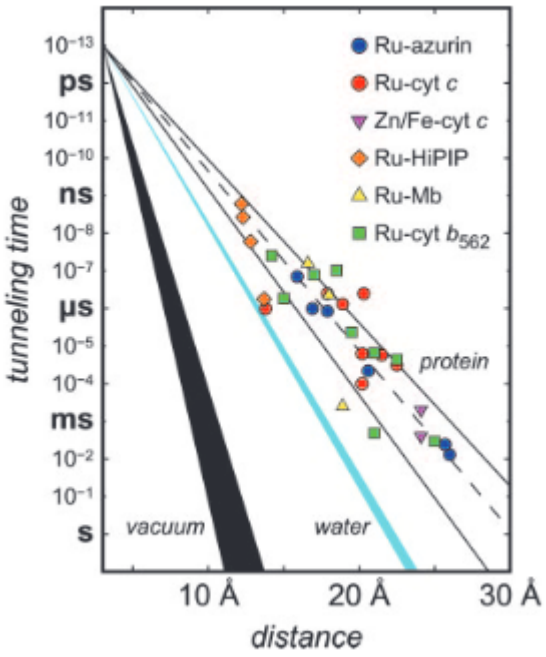
from Long-range electron transfer and Theory and Electrochemistry of Cytochrome c we can model the complicated action of the electron transfering to the next redox cofactor with a simplified form for the distance dependance of

$$|H_{AB}| \propto exp(-\frac{\gamma(p)}{2}R)$$

where p is the percent of protein media between the redox cofactors as shown in the distance prefactor graph.

Gamma calculations

The tunnel time between redox cofactors is strongly dependant on which media carries the electron as show in the graph from Long-range electron transfer



While most proteins are solid, this construct has a number of jumps that travel through the water environment. In order to deal with the different media, the path that the electron travels is examined to determine whether it moves through protein media or through the water environment.

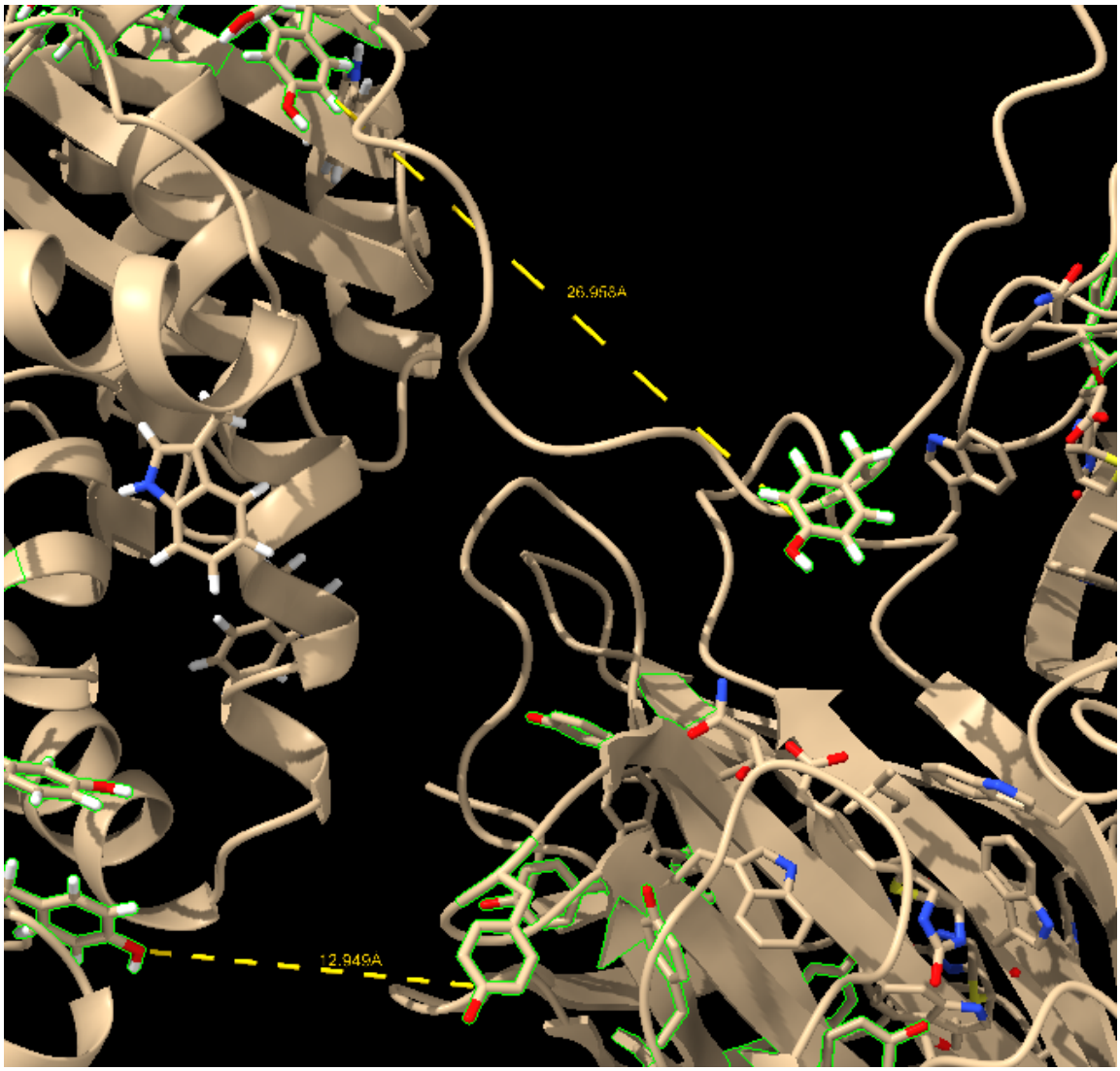
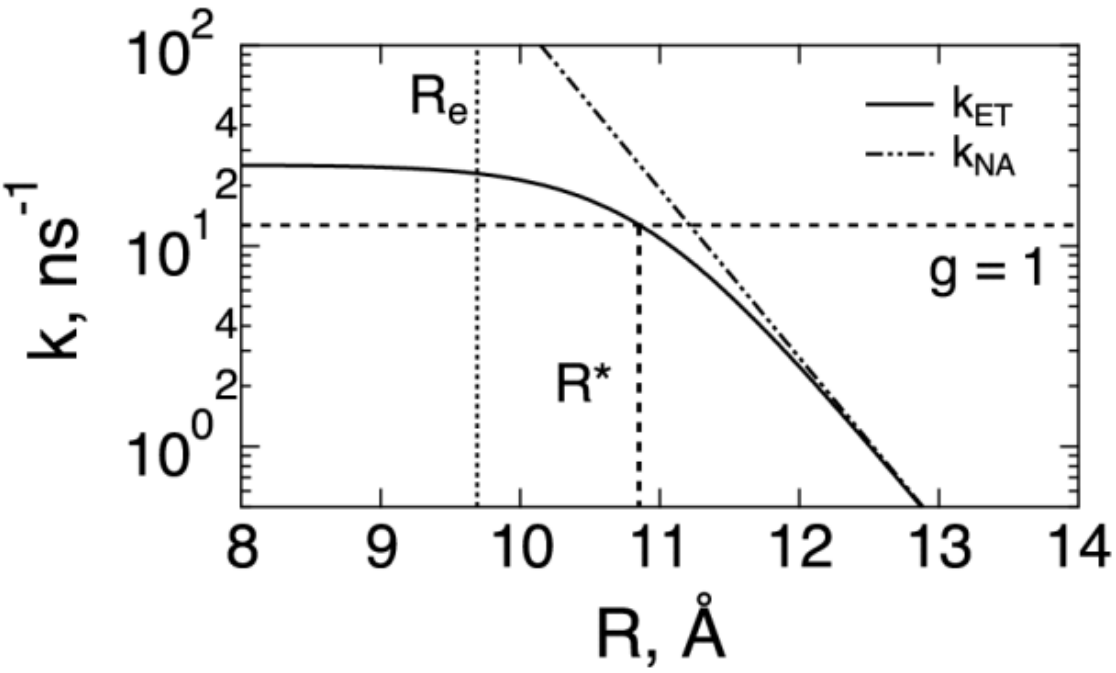


Image showing the jumps from one Tyrosine to the next and the path the electrode must travel. The gamma for each pair of neighbors is estimated to allow rate calculations to be adjusted.

Published network

We do not have the full power of an all atom simulation of the protein. Instead the distance rate parameter is estimated by interpolating the values (published) [https://pubs.acs.org/doi/10.1021/acsomega.3c02719]. We select an interaction radius, and then limit the rates at that value. As can be seen in the figure above (Distance Prefactor graph), we are able to largely mimic the given values



Static network

The values estimated from the PBS are used with Marcus theory to give jumps between the redox cofactors.

$$|H_{AB}| \propto exp(-\frac{\gamma(p)}{2}R)$$

where p is the percent of protein media between the redox cofactors as shown in the distance prefactor graph.

Vibrate network

We are assuming that the jumps performed between each redox cofact can be modeled by Marcus theory with each jump modeled by the following equation

$$k_{et} = \frac{2\pi}{\hbar}|H_{AB}|^2 \frac{1}{\sqrt{4\pi\lambda k_B T}} \exp(-\frac{\Delta G^{\dagger}}{k_B T})$$

where

$\Delta F^{\dagger} = \frac{(\lambda + \Delta G^{\circ} + e\Delta V)^2}{4\lambda k_B T}$ as shown in the Gibb's change in the graph above

from Long-range electron transfer and Theory and Electrochemistry of Cytochrome c we can model the complicated action of the electron transfering to the next redox cofactor with a simplified form for the distance dependance of

$$|H_{AB}|^2 \propto exp(-1.4R)exp(\frac{1}{2}\gamma^2 < (\delta R)^2 >)$$

last the rates are calculated by using

$$k = \frac{K_{NA}}{1+g}$$

where

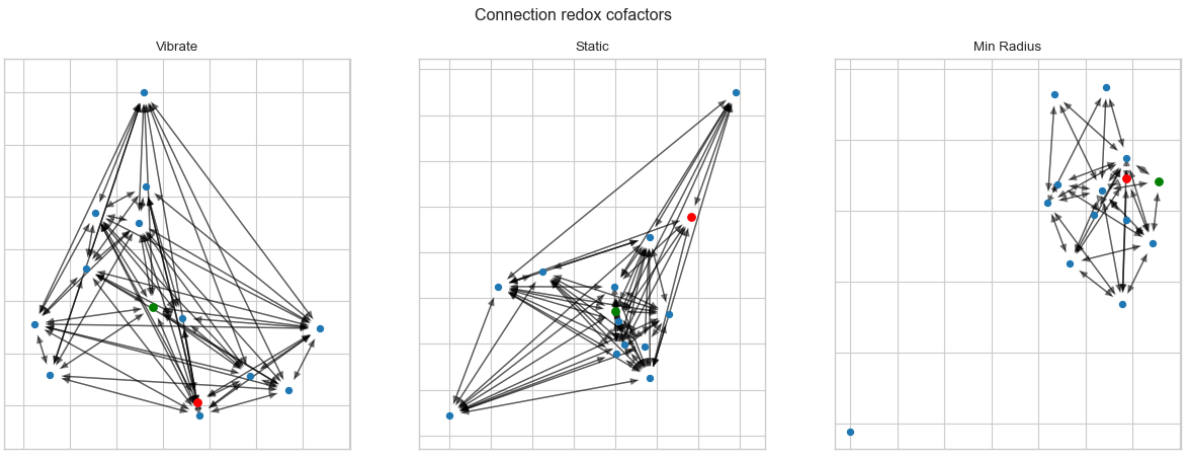
$$g = \frac{\pi V_{DA}^2 \tau_x}{\hbar\sqrt{\lambda F_{act}}} \exp \frac{3}{2}\gamma^2 < (\delta R)^2 >$$

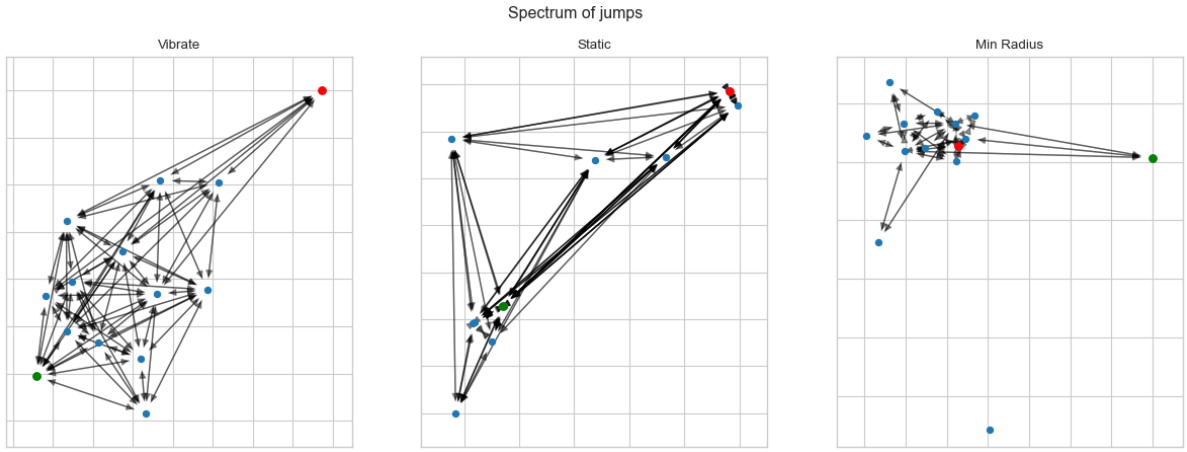
τ_x is estimated from the paper

Node maps.

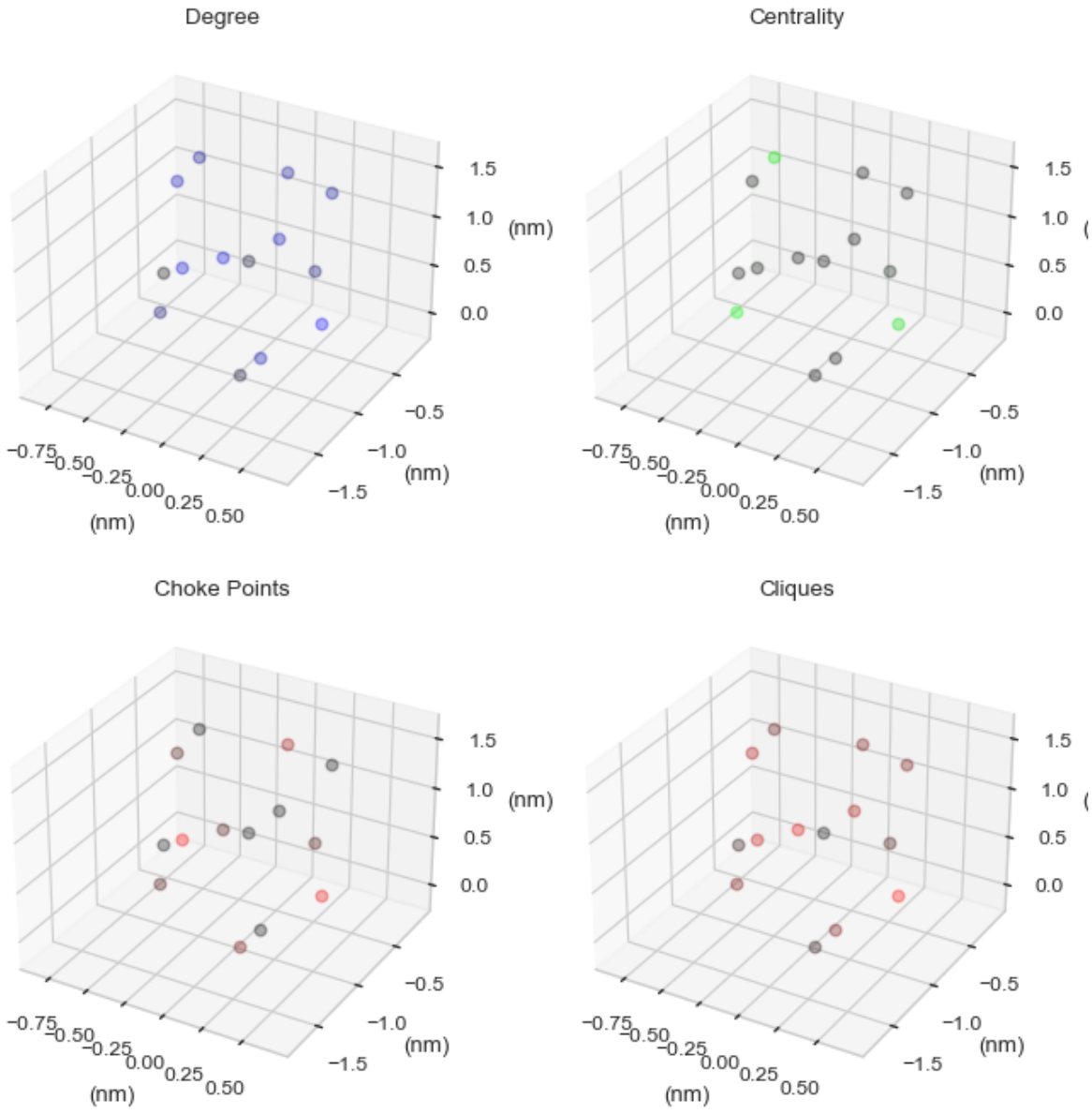
Node maps abandon the euclidean geometry completely to form networks based on the nfearest neighbor connections and the transfer times between the redox cofactors. The networks indicate that while we do have error in the various jumps, the overal random nature of the errors and the large number of interconnections between the aromatic amino acids largely cancel out to give an average value of tranferance that should match with the higher resolution all atom calculations. This is not true for the Phi29 calculations where there are single amino acids that handle almost all the current that is transfered

In [4]:
```
###############################################################################
#################      Network Visualizations         ##########################
###############################################################################


if True:
    graphs = [['Vibrate',G_vibrate],['Static', G_static],['Min Radius', G_min]]
    KMC_Plotting.DrawLayout(graphs, lambda G: nx.spring_layout(G,weight='time'), 'C
    #DrawLayout(graphs, lambda G: nx.spectral_layout(G,weight='time'), 'Spectrum of
    KMC_Plotting.DrawLayout(graphs, lambda G: nx.kamada_kawai_layout(G,weight='time
```





In [5]:
```
##############################################################################
##################      Network Analysis 2            ########################
##############################################################################

popularity= KMC_Plotting. PlotNetworkMetrics(G_connected, activeAminos)
```



Network analysis

These graphs are related only the the network of neighbors formed from these proteins. The graphs are designed to identify properties of the networks and identify those cofactors that are uniquely connected to make them important for the proteins.

Clusters

Idenify possible number of triangles through each point, allowing for communication with that node

Choke Points

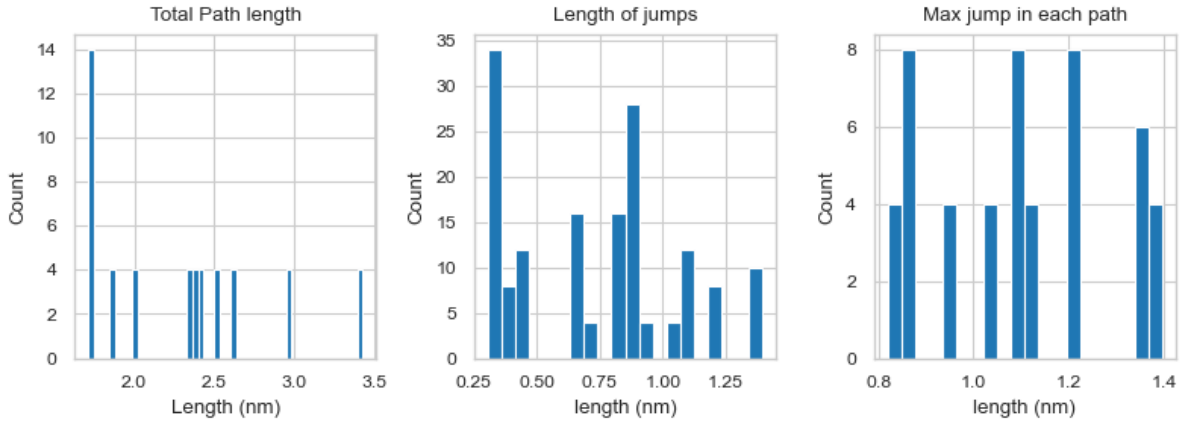
Between centrality finds all the shortest paths between all the pairs of nodes in the graph. Those nodes that participate in the most paths, are those that are likely to be the most important for conductivity. Those nodes that do not have many parallel paths are likely to be choke points and determine the conductivity of the whole construct. (wiki) [https://en.wikipedia.org/wiki/Betweenness_centrality]

Cliques

Cliques of a graph are subgraphs where any two members of a clique share a vertex. By overlaying the cliques and adding a count to a node for each overlap, we can identify those nodes that participate most in the communication through the network and are ikely important for conduction. (wiki)[https://en.wikipedia.org/wiki/Clique_(graph_theory)]

In [6]:
```
def CalculateDistancePaths(useMinimum=True):
    if useMinimum:
        weight='minDist'
    else:
        weight='distance'
    shortestPaths_connected=[]

    for startNode in [amino['aminoIndex'] for amino in injectionAminos]:
        for endNode in [amino['aminoIndex'] for amino in exitAminos]:
            shortestPaths_connected.extend( KMC_Protein. CalculatePaths(G_connected
    KMC_Plotting. PlotDistancePaths(G_connected, shortestPaths_connected, weight)

CalculateDistancePaths()
```

```
Total paths tested:  50
The average tunnel distance is 0.73 nm
All paths must make at least one jump >= 0.82 nm jump
```



Path lengths

We do a quick check to see what the greatest jump is that the electron must make with respect to distance to get from from the inject to the exit site. We then take the minimum of the longest jumps that all the paths must make to determine if there is a bottleneck that is larger than the Dutton radius.

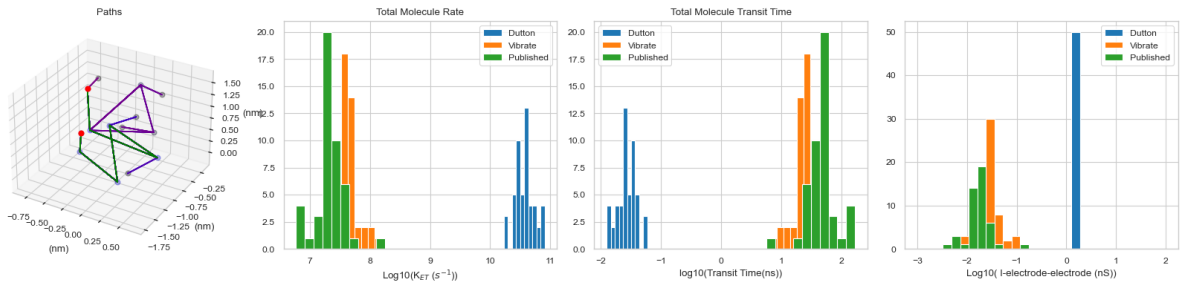
In [7]:
```
#estimate the times for the injection and exit from the electrode into the amino ac
nu=appliedVoltage_V/2
rF=  scipy.special.erfc( (injectionReorg_EV+nu)/np.sqrt( 4* kbT_eV *injectionReorg_
rB = scipy.special.erfc( (injectionReorg_EV-nu)/np.sqrt( 4* kbT_eV *injectionReorg_
print(f"{injectionEnergy/hbar_eV*rF*1e-9:.2f} 1/ns,{injectionEnergy/hbar_eV*rB*1e-9
```

```
0.03 1/ns,3.87 1/ns ,3.07e+00 nS @ 0.20 V
```

In [8]:
```
#add information about rates to the network to allow for rate calculations
KMC_Protein. RateNetwork(G_vibrate)
KMC_Protein. RateNetwork(G_static)
KMC_Protein. RateNetwork(G_min)

def AddInjectionTimes(shortest_Paths):
    nu=appliedVoltage_V/2
    rF= injectionEnergy/hbar_eV*scipy.special.erfc( (injectionReorg_EV+nu)/np.sqrt(
    rB = injectionEnergy/hbar_eV*scipy.special.erfc( (injectionReorg_EV-nu)/np.sqrt
    for path in shortest_Paths:
        path['injectionTime']=2/np.abs(rF-rB)
        path['injectPre']=electronCharge  /appliedVoltage_V



shortestPaths_static=[]
shortestPaths_vibrate=[]
shortestPaths_min=[]

for startNode in [amino['aminoIndex'] for amino in injectionAminos]:
    for endNode in [amino['aminoIndex'] for amino in exitAminos]:
        shortestPaths_static.extend( KMC_Protein. CalculatePaths(G_static, activeAm
        shortestPaths_vibrate.extend(  KMC_Protein. CalculatePaths(G_vibrate , acti
        shortestPaths_min.extend(  KMC_Protein. CalculatePaths(G_min , activeAminos
```

```
AddInjectionTimes(shortestPaths_static)
AddInjectionTimes(shortestPaths_vibrate)
AddInjectionTimes(shortestPaths_min)

shortestPaths_static=KMC_Protein.sortPaths(shortestPaths_static, 'time')
shortestPaths_vibrate=KMC_Protein.sortPaths(shortestPaths_vibrate, 'time')
shortestPaths_min=KMC_Protein.sortPaths(shortestPaths_min, 'time')


KMC_Plotting. PlotGraphPaths(activeAminos,
                      shortestPaths_static,shortestPaths_vibrate,shortestPaths_min
                      G_static, G_vibrate, G_min,
                      injectionAminos,exitAminos)
```

```
Target 11 cannot be reached from given sources : 11 : TYR 82
injectTimes:5.21e-10 , injectPre:8.01e-19, ave transite time:2.89e-11
traveltime:5.50e-10 , current:1.46e-09
Network :  Dutton
Total paths tested:  50
The average travel time is 2.33e-02 ns
The molecule K(ET) is 4.28e+10 1/s



injectTimes:5.21e-10 , injectPre:8.01e-19, ave transite time:2.79e-08
traveltime:2.84e-08 , current:2.82e-11
Network :  Vibrate
Total paths tested:  50
The average travel time is 2.35e+01 ns
The molecule K(ET) is 4.25e+07 1/s



injectTimes:5.21e-10 , injectPre:8.01e-19, ave transite time:5.38e-08
traveltime:5.43e-08 , current:1.47e-11
Network :  Published
Total paths tested:  46
The average travel time is 4.34e+01 ns
The molecule K(ET) is 2.31e+07 1/s
```



Shortest Paths

Finding the shortest paths from our injection to exit sites allows the simulation to find the upper bounds of the transfer rate

In order to get statistics for the paths, Dijkstra and Bellman-Ford's methods are used to find the shortest paths with respect to the tranfer time between the nodes.

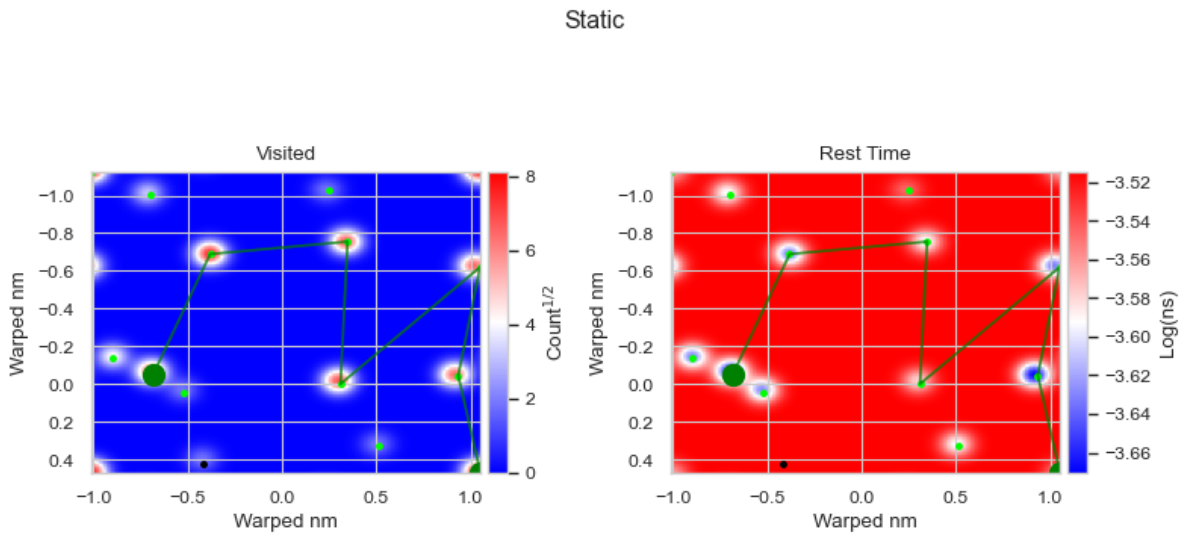
Additional information is gathered by requiring the Dijkstra and Bellman-Ford methods to visit random nodes in the molecule, resulting in a statisical variation of the routes and possible times.

Once the times have been identified, the transit time for the molecule is calculated by summing the times for each jump and then taking an inverse.

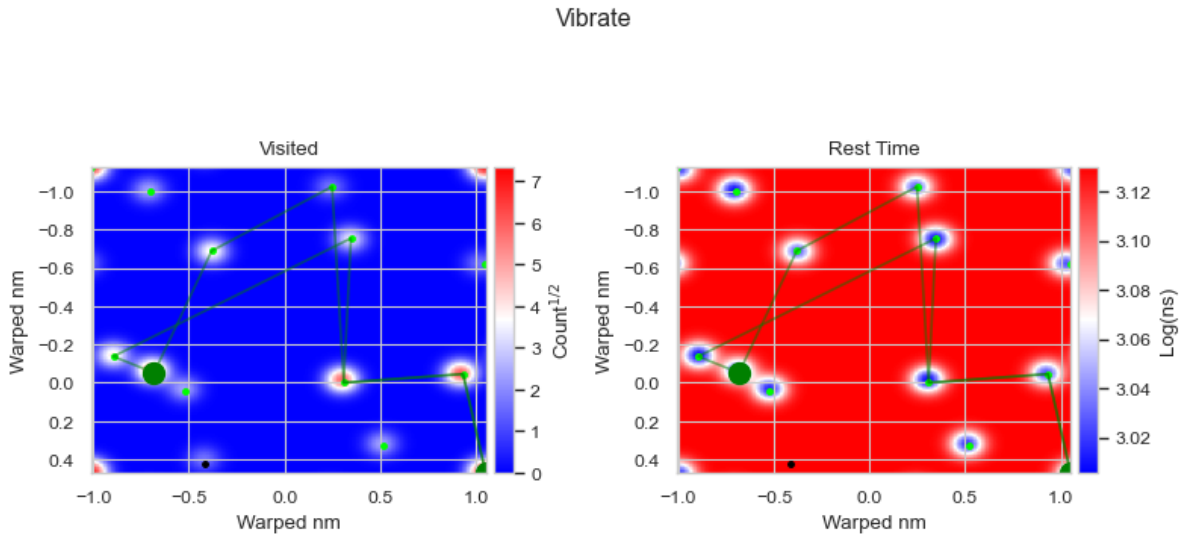
The current is estimated from the rate, ignoring the contact resistance and other factors that may play a role in that calculations

In [ ]:
```
KMC_Plotting.PlotPathMetrics(G_static, activeAminos,shortestPaths_static, "Static")
KMC_Plotting.PlotPathMetrics(G_vibrate, activeAminos,shortestPaths_vibrate, "Vibrat
KMC_Plotting.PlotPathMetrics(G_min, activeAminos,shortestPaths_min, "Published")

del  shortestPaths_static, shortestPaths_vibrate, shortestPaths_min
```

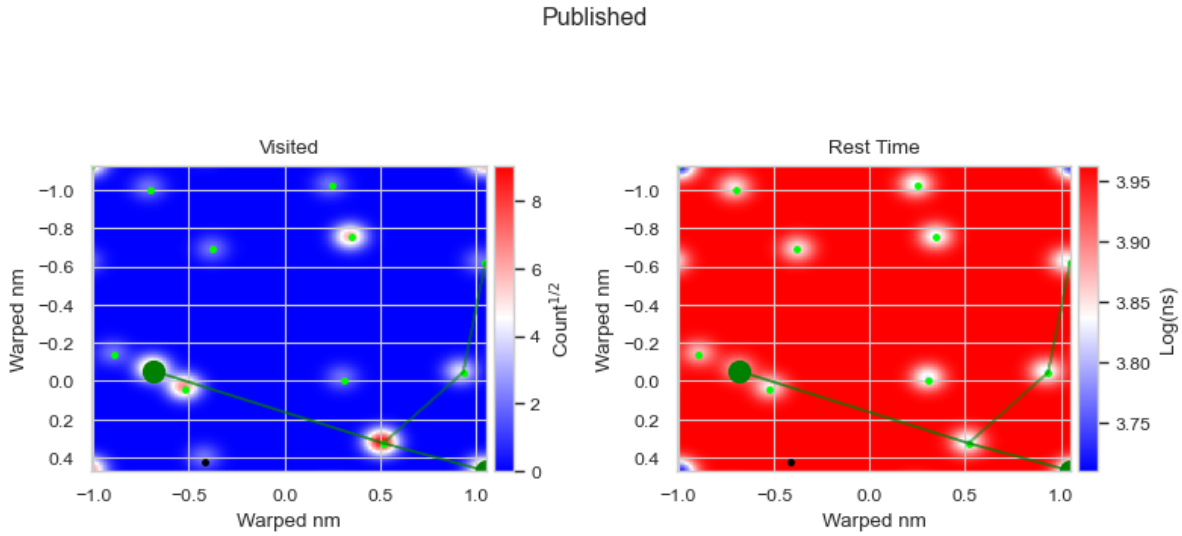
```
Smoothing Energy Map
```




```
Smoothing Energy Map
```



Smoothing Energy Map



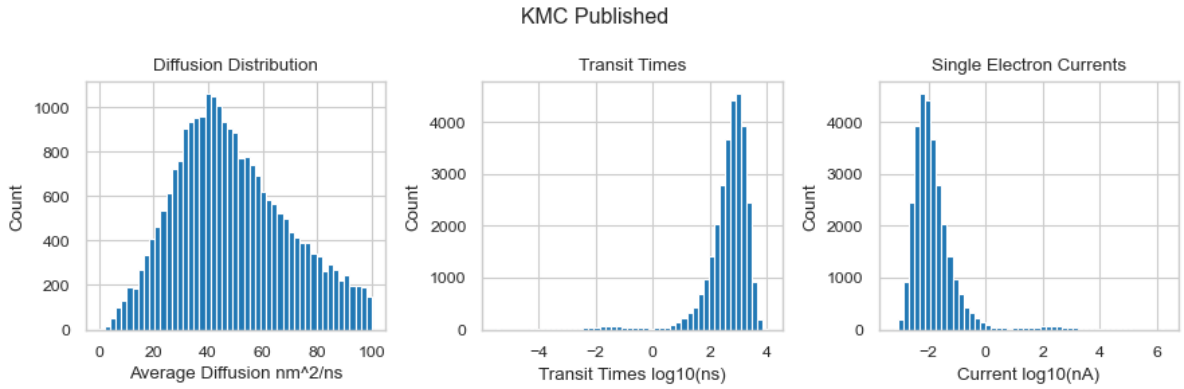
Rates Graphs

Once again the maniform transform is used to show the pathways through the system. For the case of the rest time on the left, the tranfer rates inside the proteins show rapid exchange and diffusion within each protein, but very slow rates between the center. The dots show the locations of each of the redox cofactors and the shade of green indicates that redox energy needed to oxidize that molecule. Frequently, the path does jump through the harder to react Met residues, which may not be physical in an actual protein

The visited graph shows that some sites on the protein are virtually required for the graph to be able to reach the exit. These sites are critical and are in the most mobile area of this construct, resulting in unstable transit times.

In [ ]:
```
##############################################################################
#################      Kinetic Monti Carlo          ##########################
##############################################################################
```

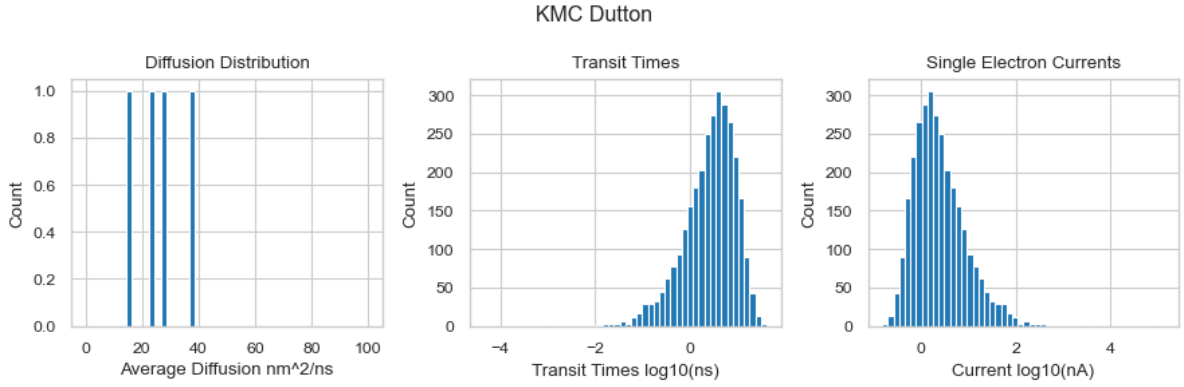
```
successDwellTimes, dwellTimes, passes, electronTimes, diffusions=KMC_Protein.KMC(G_
KMC_Plotting.PlotKMC(  electronTimes, diffusions,'KMC Published')
```

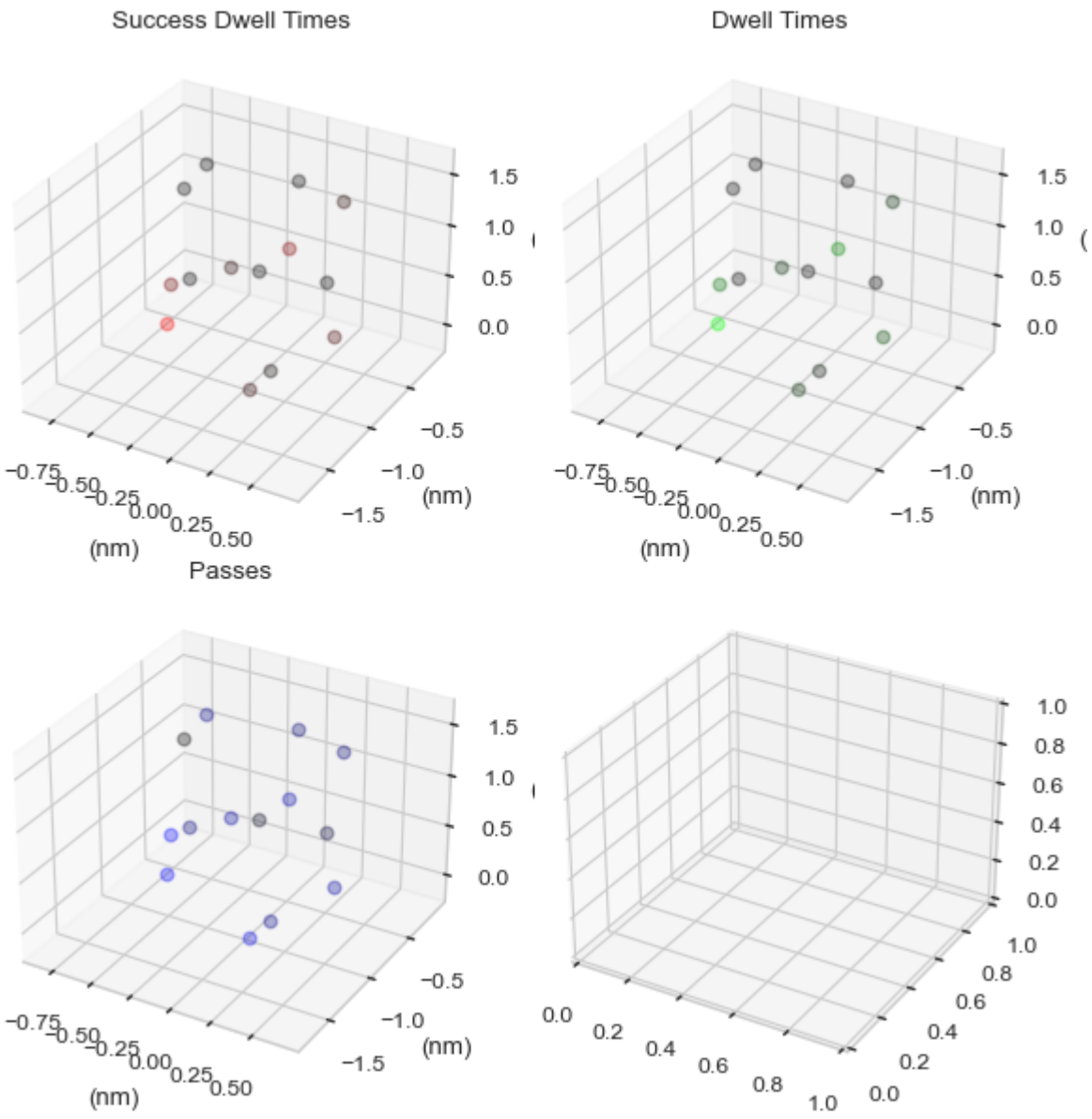


Comparison with the published values

The overall rates compare favorable with those in the publication for this model (expected value of 20 nm^2/ns). We still get what would be very low currents from the model, especially when the resistance of the contacts is also added to this system. A better model would allow multiple electrons to exist inside the molecule and estimate the injection rate with added electrons.

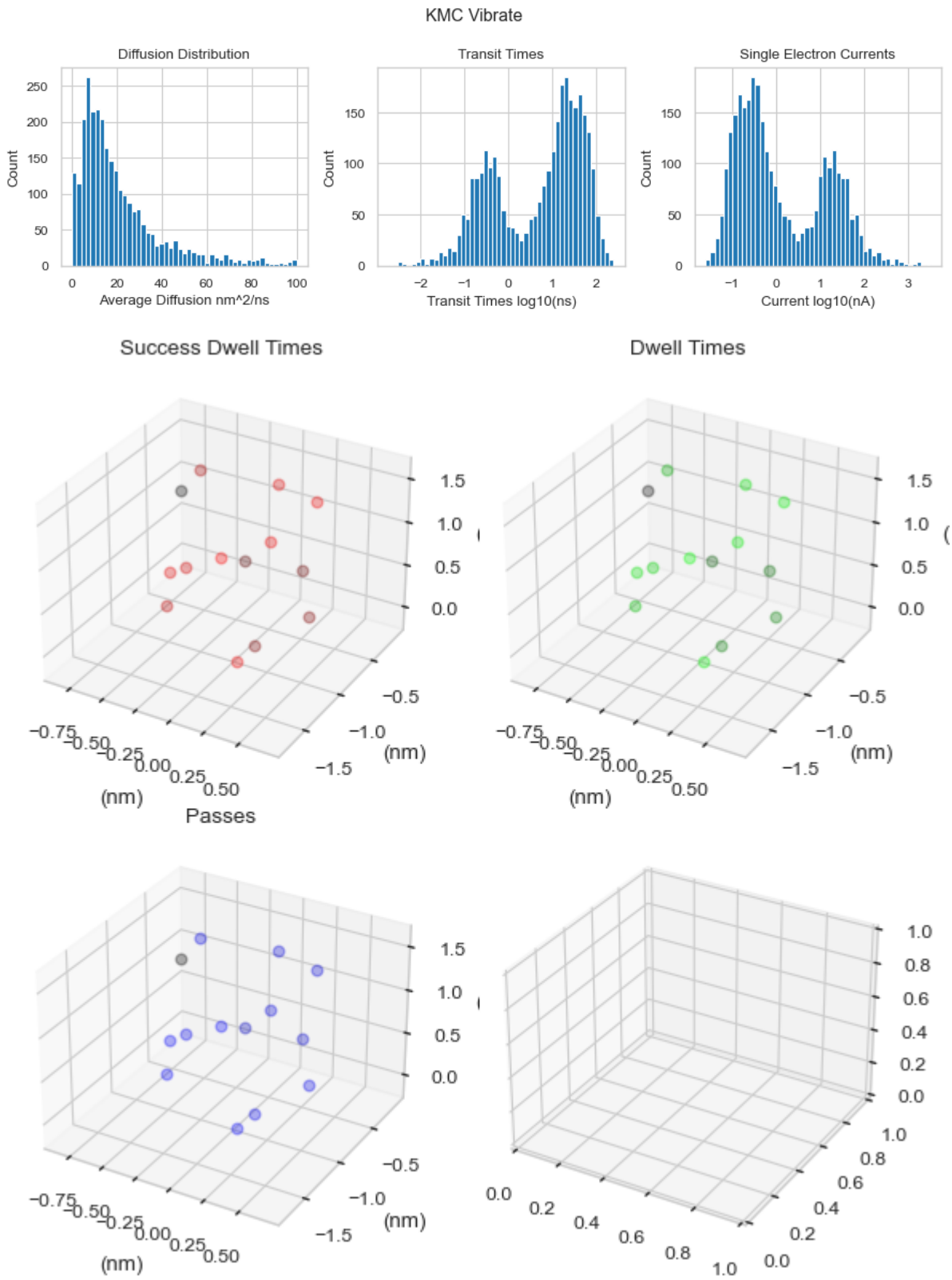
In [ ]:
```
successDwellTimes, dwellTimes, passes, electronTimes, diffusions=KMC_Protein. KMC(G
KMC_Plotting. PlotKMC(  electronTimes, diffusions,'KMC Dutton')
KMC_Plotting.PlotKMCNetworks(activeAminos,successDwellTimes, dwellTimes, passes)
```





In [ ]:

```
successDwellTimes, dwellTimes, passes, electronTimes, diffusions=KMC_Protein.KMC(G_
KMC_Plotting. PlotKMC(  electronTimes, diffusions,'KMC Vibrate')

KMC_Plotting.PlotKMCNetworks(activeAminos,successDwellTimes, dwellTimes, passes)
```



In [ ]:
```
def MoveElectron(G_test, electronLocation):
    """Walk through the probabilities avaible at this node and select the node base
    The nodes have been arranged from most probably to least to allow a quick looku

    Args:
        G_test (graph): graph that has the exit probabilities for each node availab
        electronLocation ( int ): current node location of electron
```

```
    Returns:
        _type_: next node location,                time to move to next node
    """
    newLocation=-1
    rate =0
    #get all the options for electron at this step
    rates = G_test.nodes[electronLocation]['rates']

    v=rng.random()
    for i in range(len(rates)):
        if v<rates[i]:
            newLocation = G_test.nodes[electronLocation]['targets'][i]
            jumptime = G_test.nodes[electronLocation]['times'][i]
            break

    if newLocation == -1:
        newLocation = G_test.nodes[electronLocation]['targets'][-1]

    totalRate = G_test.nodes[newLocation]['outrate']

    return newLocation, totalRate, jumptime

def KMC(G_test,activeAminos, injectionAminos,exitAminos, numberElectrons=5000, maxI
    """_summary_

    Args:
        G_test (_type_): _description_
        startElectronNode (_type_): _description_
        activeAminos (_type_): _description_
        endElectroneNode (_type_): _description_
        numberElectrons (int, optional): _description_. Defaults to 5000.
        maxIterations (int, optional): _description_. Defaults to 500000.

    Returns:
        _type_: _description_
    """

    startElectronNodes=[amino['aminoIndex'] for amino in injectionAminos]
    endElectroneNodes= [amino['aminoIndex'] for amino in exitAminos]

    successDwellTimes = np.zeros(len(activeAminos))
    dwellTimes = np.zeros(len(activeAminos))
    passes = np.zeros(len(activeAminos))
    electronTimes =[]
    diffusions =[]

    locations = np.zeros(maxIterations, dtype=int)
    distances = np.zeros(maxIterations, dtype=float)
    times = np.zeros(maxIterations, dtype=float)

    for attempt in range(numberElectrons):

        #choose a random start and end point
        electronLocation=  startElectronNodes[rng.integers(0,len(startElectronNodes
        seeking = True
```

```
        cc=-1

        while seeking:
            cc+=1

            #look at the rates to get the next location
            newLocation, Q , jumptime=MoveElectron(G_test, electronLocation)

            #move the time forward based on the rates available
            timeStep = 1/Q*np.log(1/rng.random())

            #get the distance that will be jumped
            dx=G_test[electronLocation][newLocation]['minDist']
            #dx= np.linalg.norm( activeAminos[electronLocation]['centerOfMass']-act

            #record the locations and rates
            locations[cc] = electronLocation
            distances[cc] =  dx
            times[cc] = timeStep

            #mark the current location with how long the electron stays there
            dwellTimes[electronLocation] += timeStep
            #mark that the electron has passed through this location
            passes[electronLocation] += 1

            #move the electron to the next location
            electronLocation = newLocation

            #if the electron has reached the injection node, record the dwell times
            if (cc>=maxIterations-1):
                seeking = False
                diffusions.append(np.mean( (distances[:cc]**2)/(2*times[:cc])))
                cc=-1
            #check if we have found the endpoint
            elif electronLocation in endElectroneNodes:
                seeking = False
                for i in range(cc):
                    successDwellTimes[locations[i]] += times[i]

                diffusions.append(np.mean( (distances[:cc]**2)/(2*times[:cc])))
                electronTimes.append(np.sum( times[:cc]))
                cc=-1

    return successDwellTimes, dwellTimes, passes, electronTimes, diffusions
```