Supplementary Information

Contents

Table S1: Fitting discussion

To fit the collected 39 data to a model, a fitting algorithm is required. This study used lmfit: a curve fitter for Python that uses the Levenberg-Marquardt method. Iteratively adjusting a model's parameters, the fitting method accepts modifications that move the model closer to the data being fitted and rejects deviations from it. This approach is repeated until the algorithm cannot enhance the model fit to the data. The method's inherent flaw is that it doesn't always yield the best fit. For instance, if one parameter deviates much from the true value while the others approach it closely, adjusting the far-parameter could first make the fit worse before getting better. Local vs global minima is the term for this concept, where a minima is an idealised collection of fitting parameters. A local minima denotes the best fit within a limited parameter space, whereas a global minimum reflects the best overall fit throughout the entire parameter space. To avoid local minima, boundaries and initialisation are used. Boundaries refer to the hard limits on the possible values that a given fitting parameter can be. For example, Φw must be between 0 and 1, since these numbers represent a system with 0% and 100% water, respectively. Values less than 0 or greater than 1 would be nonsensical as they describe states that do not exist physically. By limiting the parameter space to values closer to the expected values, the likelihood of optimising into a local minima is reduced. The initial values for each parameter, from which the algorithm iteratively alters them, are referred to as initialisation. The possibility of a global minima fit increases if the initial values are near the global minimal value. Ideally, by fitting the same data several times with various initialisation values, one may compare the fits and narrow down to the best fit, increasing the chance of finding the global minima. However, the computational time necessary for this type of robust initialisation is beyond the scope of this work. Rather, the initialisation was estimated using the value that was most likely to occur in practice. When lmfit optimises the parameters for a dataset, it also returns the standard error for the fit. This is illustrated graphically by the error bars in this study. Lmfit calculates the standard error using a covariance matrix, which can fail when the fitted value is close to the bounds. As a result, any fitted parameters without a determined standard error were excluded. Finally, only fits with an r-squared value exceeding 0.98 were considered.

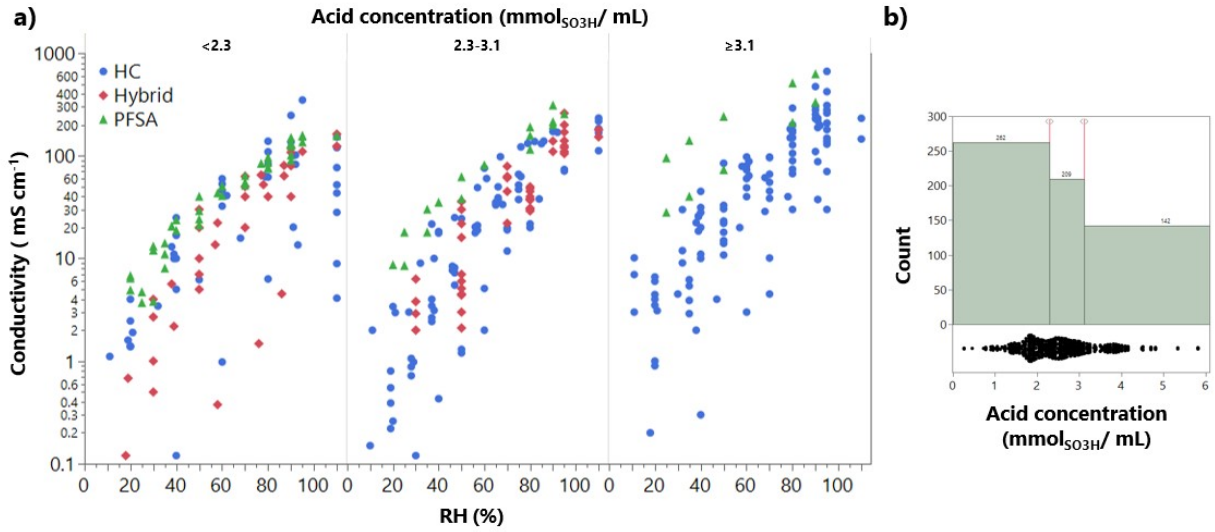
Parameter	Initialisation	Minimum Value	Maximum Value
Φ_c	0.1	0.001	1
D_p	1×10^{-9}	0	minimum diffusion value*
D_w	1×10^{-4}	1×10^{-5}	1×10^{-3}
t	0.88	0.2	2

Script: Fitting data Python Script

```
# -*- coding: utf-8 -*-
"""
General effective medium equation fitter


"""

#imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from lmfit import Model

#models
def GEM(x, crit, Dpoly, Dwater, t):
    Phipoly = (((Dpoly**(1/t))-(x**(1/t))))/(((Dpoly**(1/t))+(((1-crit)/crit)*(x**(1/t))))
    Phiwater = (((Dwater**(1/t))-(x**(1/t))))/(((Dwater**(1/t))+(((1-crit)/crit)*(x**(1/t))))
    return((-1*Phipoly)/(Phiwater-Phipoly))

def Maxconc(wfrac, acidconc):
    a = (972/18.01528)*wfrac
    b = acidconc*(1-wfrac)
    if a>b:
        return(b)
    else:
        return(a)

def Dcalculator(conductivity, pconc):
    return(((conductivity/1000)*8.314*353.15)/((96485.3321**2)*(pconc/1000)))

def WSLang(x, A, bL):
    return (A*bL*x)/(1+(bL*x))
def WSHenr(x, s):
    return s*x
def WSClus(x, Kc, n):
    return n*(Kc*(x**n))
def Park_Model(x, A, bL, s, Kc, n):
    return ((A*bL*x)/(1+(bL*x)))+(s*x)+((Kc*n*(x**n)))

def find_nearest(array, value):
    array = np.asarray(array)
    idx = (np.abs(array - value)).argmin()
    return array[idx]

def tortuosity(x, D, t):
    return ((D*x)/t)

#modelling
GEM_array = []
graphing_array = []
park_array = []

file = r"J:\electrocat\William Bangay\Lit review\fitting data.xlsx"
data = pd.read_excel(file, sheet_name="80C")


cite_key = data['Citation Key'].drop_duplicates()
for z in range(len(cite_key)):
    lit_name = data["Name in literature"][data["Citation Key"] == cite_key.iloc[z]].drop_duplicates()
    for y in range(len(lit_name)):
        single_set = (data.loc[(data["Name in literature"] == lit_name.iloc[y]) & (data["Citation Key"] == cite_key.iloc[z])])
        if len(single_set) > 3:
            waterfraction = single_set['Water volume fraction'].to_numpy()
            dryacidconcentration = single_set['Dry [SO3]'].to_numpy()
            conductivity = single_set['Conductivity'].to_numpy()
            polymerclass = single_set['Polymer class'].to_numpy()
            relhumidity = single_set['RH'].to_numpy()
            lamb = single_set['lambda'].to_numpy()
            Philic_Phobic_ratio = single_set['Molar Ratio of philic to phobic groups'].to_numpy()

            protonconc = []
            x = []
            for w in range(len(waterfraction)):
                protonconc = np.append(protonconc, Maxconc(waterfraction[w], dryacidconcentration[w]))
                x = np.append(x, Dcalculator(conductivity[w], protonconc[w]))

            mod = Model(GEM)
            pars = mod.make_params(crit={'value': 0.1, 'min':0.001, 'max':1},
                                   Dpoly={'value': 0.000000001, 'min':0, 'max':np.min(x)},
                                   Dwater={'value': 0.0001, 'min': 0.00001, 'max': 0.001, 'vary': True},
                                   t={'value':0.88, 'min':0.2, 'max': 2, 'vary':True})
            GEM_result = mod.fit(waterfraction, pars, x=x, nan_policy='propagate')

            y_spread = np.linspace(np.min(x)/10, np.max(x)*10, 10000)
            crit, Dpoly, Dwater, t = GEM_result.params['crit'], GEM_result.params['Dpoly'], GEM_result.params['Dwater'], GEM_result.params['t']

            plt.figure(0)
            plt.plot(waterfraction, x, 'o', label='Raw')
            plt.plot(GEM(y_spread, crit, Dpoly, Dwater, t), y_spread, '--', label='Fit')
            plt.xlabel("Water volume fraction")
            plt.ylabel("D")
            plt.yscale("log")
            plt.xlim(0,1)
            plt.title(str(lit_name.iloc[y]) + ' from ' + str(cite_key.iloc[z]))
            plt.legend()
            plt.show()

            #tortuosity
            mod = Model(tortuosity)
            pars = mod.make_params(D={'value': 0.0034, 'min':0, 'vary':True},
                                   t={'value':2, 'min':1, 'vary':True})
            tort_result = mod.fit(x, pars, x=x, nan_policy='propagate')

            y_waterfrac = np.linspace(0,1,1000)
            D, tort = tort_result.params['D'], tort_result.params['t']
```

```
        plt.figure(1)
        plt.plot(waterfraction, x, 'o', label='Raw')
        plt.plot(y_waterfrac, tortuosity(y_waterfrac, D, tort), '--', label='Fit')
        plt.xlabel("Water volume fraction")
        plt.ylabel("D")
        plt.yscale("log")
        plt.xlim(0,1)
        plt.title(str(lit_name.iloc[y]) + ' from ' + str(cite_key.iloc[z]))
        plt.legend()
        plt.show()

        #critical lambda
        critical_lambda = ((crit.value*(972/18.01528))/dryacidconcentration[0])
        if crit.stderr is None:
            critical_lambda_stderr = 100
        else:
            critical_lambda_stderr = ((crit.stderr*(972/18.01528))/dryacidconcentration[0])

        #Park modelling
        x = []
        for w in range(len(waterfraction)):
            x = np.append(x, relhumidity[w]/100)

        mod = Model(WSLang) + Model(WSHenr) + Model(WSClus)
        pars = mod.make_params(A={'value': 1, 'min': 0},
                               bL={'value': 100, 'min': 0, 'vary': True},
                               s={'value': 6, 'min': 0},
                               Kc={'value': 1, 'min': 0},
                               n={'value': 6, 'min': 0})

        Park_result = mod.fit(lamb, pars, x=x)

        WA = np.linspace(0,1,1000)
        comps = Park_result.eval_components()

        A, bL, s, Kc, n = Park_result.params['A'], Park_result.params['bL'], Park_result.params['s'], Park_result.params['Kc'], Park_result.params['n']

        plt.figure(2)
        plt.plot(WA, Park_Model(WA, *Park_result.params.values()), '--', label='Fit') # '*' will dump all iterables in a list
        plt.plot(WA, WSLang(WA, A, bL), '--', label='Langmuir')
        plt.plot(WA, WSHenr(WA, s), '--', label='Henry')
        plt.plot(WA, WSClus(WA, Kc, n), '--', label='Clustering')
        plt.plot(x, lamb, 'o', label='Raw')
        plt.xlim(0,1)
        plt.legend()
        plt.show()

        WA_Lambda_array = np.transpose([WA, Park_Model(WA, *Park_result.params.values())])

        closest_lambda_to_critical_percolation = find_nearest(Park_Model(WA, *Park_result.params.values()), critical_lambda)
        upper_lambda_to_critical_percolation = find_nearest(Park_Model(WA, *Park_result.params.values()), (critical_lambda+critical_lambda_stderr))
        lower_lambda_to_critical_percolation = find_nearest(Park_Model(WA, *Park_result.params.values()), (critical_lambda-critical_lambda_stderr))

        RH_at_critical_percolation = WA_Lambda_array[np.where(WA_Lambda_array[:,1] == closest_lambda_to_critical_percolation)]
        RH_at_upper = WA_Lambda_array[np.where(WA_Lambda_array[:,1] == upper_lambda_to_critical_percolation)]
        RH_at_lower = WA_Lambda_array[np.where(WA_Lambda_array[:,1] == lower_lambda_to_critical_percolation)]

        crit_RH = RH_at_critical_percolation[0,0]*100
        upper_crit_RH = RH_at_upper[0,0]*100
        lower_crit_RH = RH_at_lower[0,0]*100

        #exporter
        for w in range(len(waterfraction)):
            graphing_array = np.append(graphing_array, [cite_key.iloc[z], lit_name.iloc[y], waterfraction[w], Dcalculator(conductivity[w], protonconc[w]), polymerclass[w], Philic_Phobic_ratio[w],
                                                        crit.value, crit.stderr, Dpoly.value, Dpoly.stderr, Dwater.value, Dwater.stderr, t.value, t.stderr, GEM_result.rsquared, GEM_result.redchi, GEM_result.aic, GEM_result.bic,
                                                        crit_RH, upper_crit_RH, lower_crit_RH, dryacidconcentration[0], A.value, A.stderr, bL.value, bL.stderr, s.value, s.stderr, Kc.value, Kc.stderr, n.value, n.stderr,
Park_result.rsquared, Park_result.redchi, Park_result.aic, Park_result.bic,
                                                        D.value, D.stderr, tort.value, tort.stderr])

    else:
        ()

graphing_array = np.reshape(graphing_array, (-1, 40))
graphing_array_export = pd.DataFrame(graphing_array,
                             columns=['Citation Key', 'Name in Literature', 'Vol fraction', 'D', 'Ionomer type', 'Ratio of philic to phobic segments',
                                      'critical percolation threshold', 'critical percolation threshold Confidence Interval', 'polymer diffusion', 'polymer diffusion Confidence Interval', 'water diffusion', 'water diffusion Confidence
Interval', 't', 't Confidence Interval', 'GEM r-squared', 'GEM reduced chi-squared', 'GEM AIC', 'GEM BIC',
                                      'RH at critical percolation threshold', 'Upper critcal', 'Lower Critical RH', '[SO3H]', 'A', 'A interval', 'bL', 'bL interval', 's', 's interval', 'Kc', 'Kc interval', 'n', 'n interval', 'Park r-squared', 'Park
reduced chi-squared', 'Park AIC', 'Park BIC',
                                      'D tort value', 'D tort error', 'tort value', 'tort error'])
graphing_array_export.to_excel("Lit_fit_dataset_3_with_tort.xlsx")
```

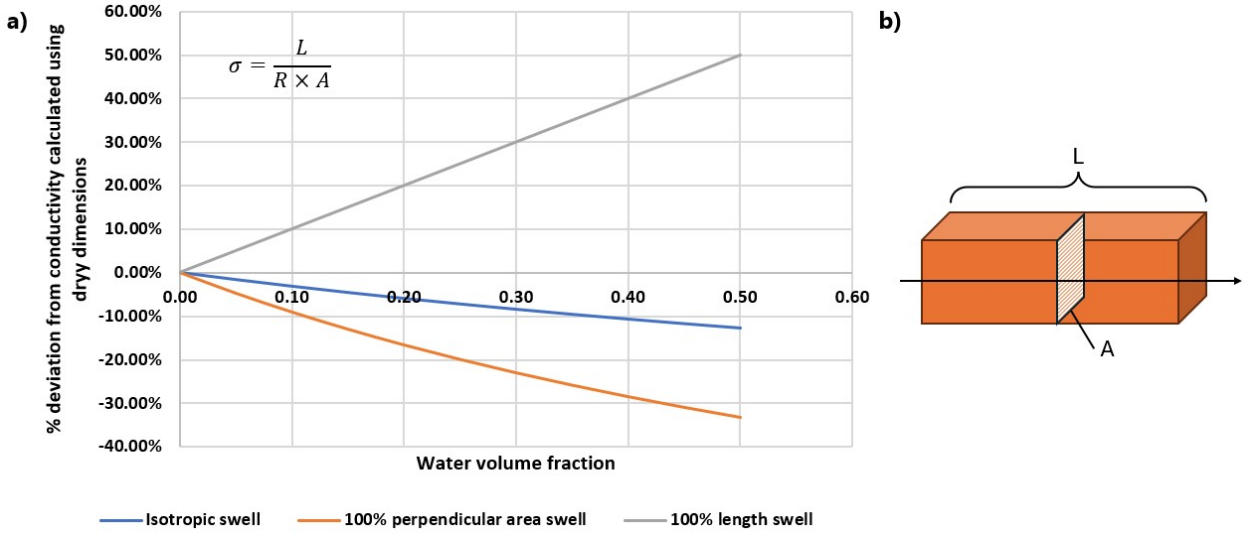
Fig. S1: Alternative version of Fig. 1



a) Conductivity at 80 °C against relative humidity, data at 110% RH represents submerged conductivity measured in water. Marker colour and shape represent the polymer type. Data split into different acid concentrations (calculated by multiplying Ion Exchange Capacity by polymer density). Data taken from references 5,7–31.

b) Representation of the data spread across the acid concentrations.

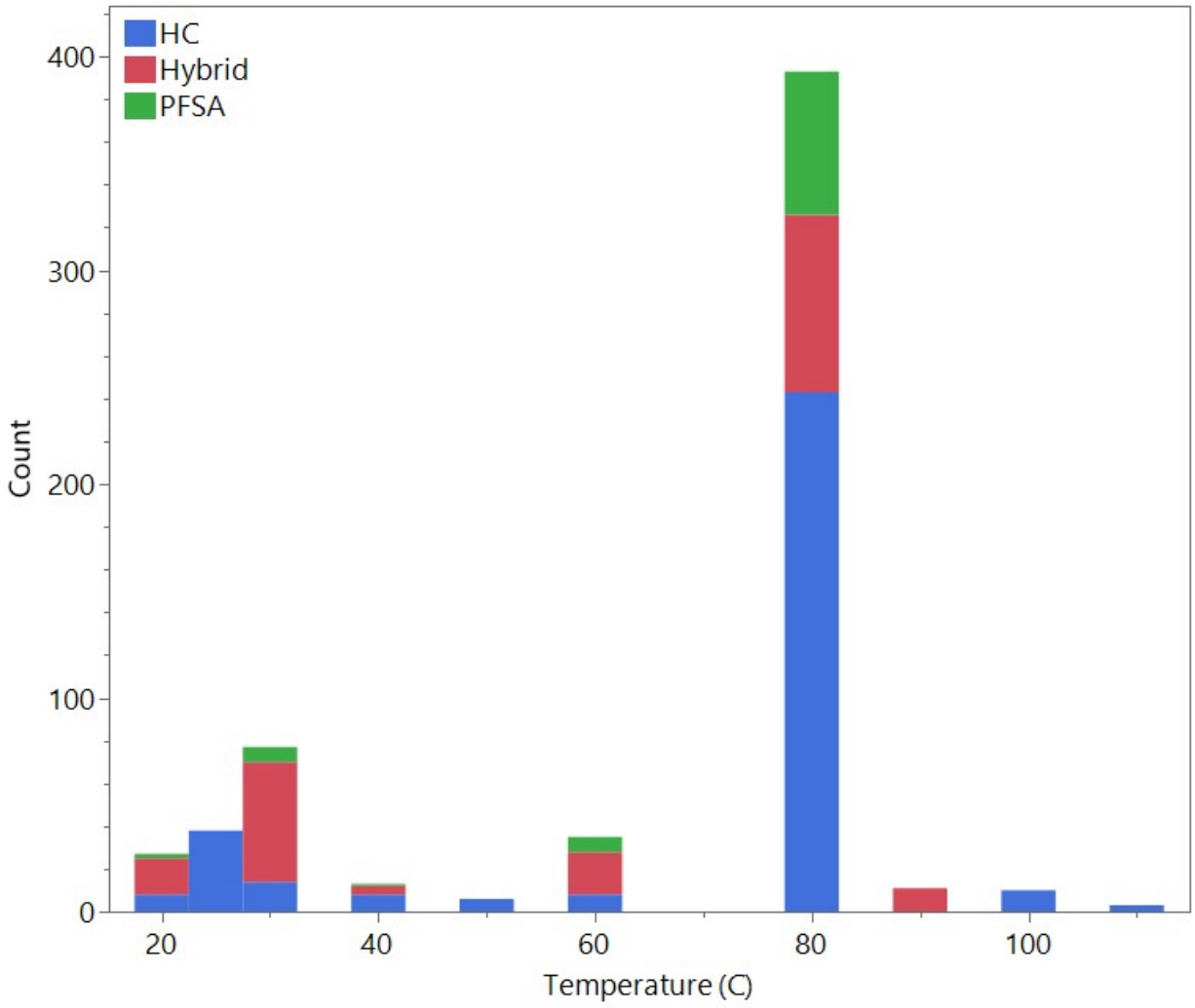
Fig. S2: Graphical representation difference between conductivity calculated using dry dimensions vs swollen dimensions.



a) Since all conductivity values collected used dry dimensions (as is the literature standard) the more a membrane swells the greater the potential deviation the true conductivity could be from the dry dimension conductivity. Inset equation is the resistance to conductivity equation.
Isotropic swell means all dimensions increase by the same amount as water volume fraction increases.
100% perpendicular swell means only the cross sectional area through which the protons travel increases as water volume fraction increases.
100% length swell means on the length (distance) that protons travel increases as the water volume fraction increases.
These results represent the maximum uncertainty.

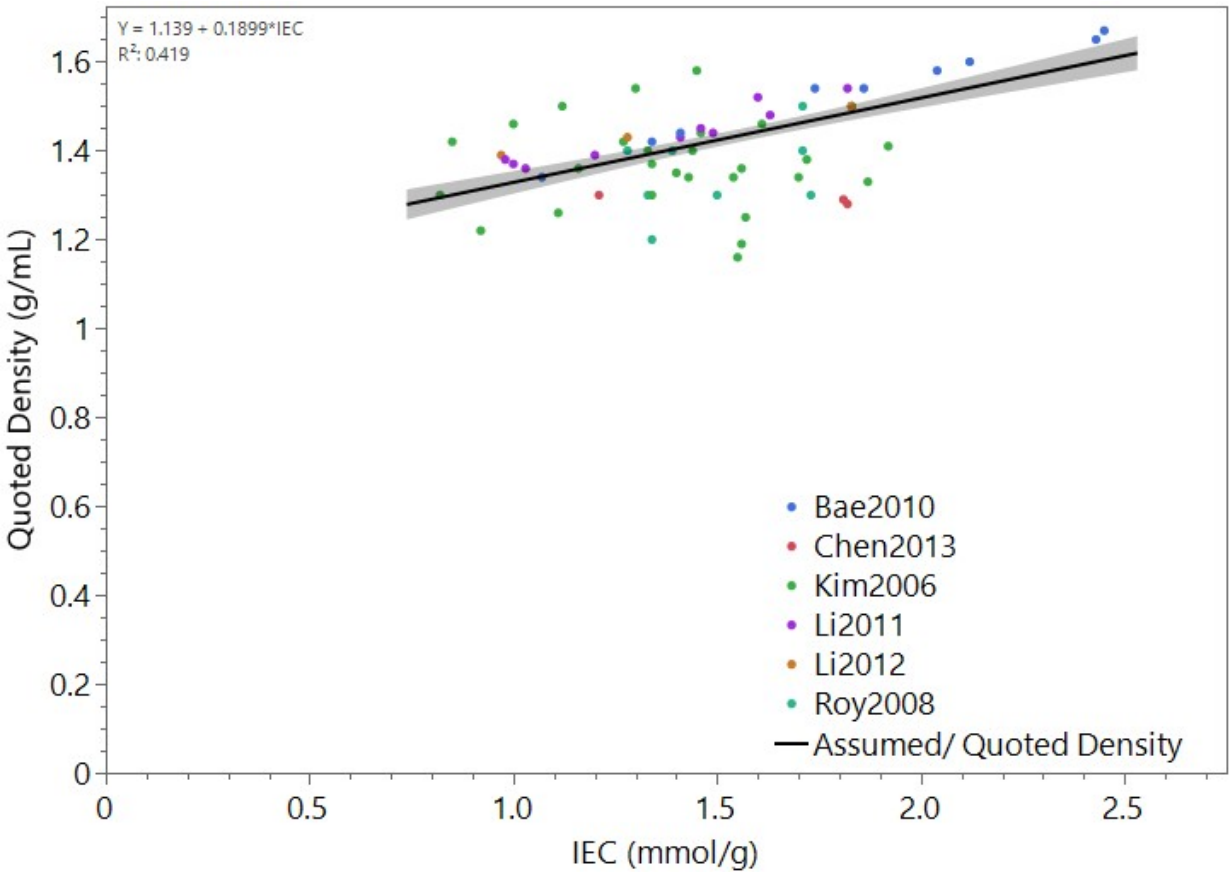
b) Example membrane. Arrow represents proton transport, L is the length of the membrane through which protons travel, and A is the perpendicular cross sectional area through which protons travel.

Fig. S3: Bar chart of all data collected at each temperature.



Data from references 7-9, 11, 12, 14, 18-21, 23-25, 28, 31, 33-38

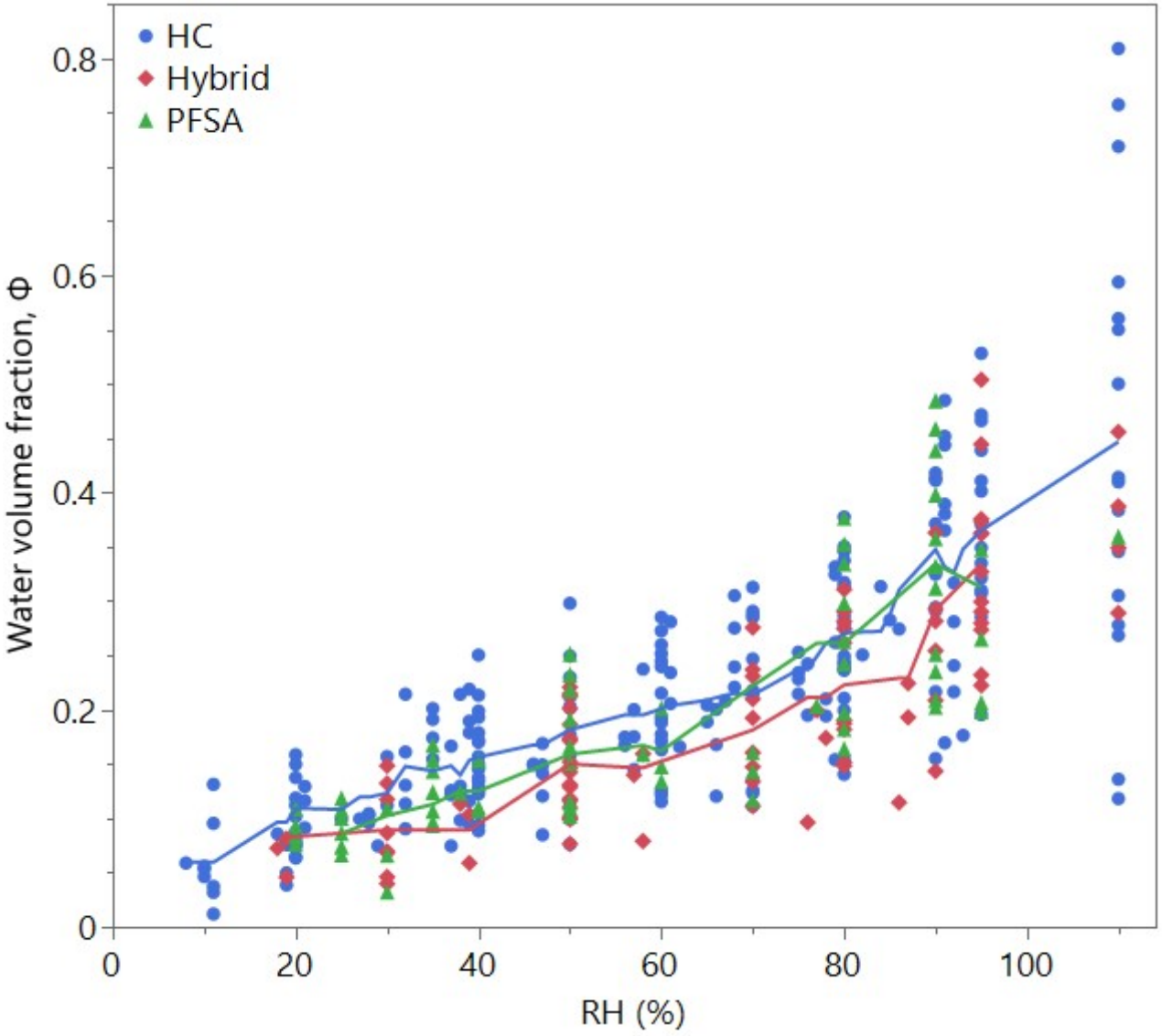
Fig. S4: Expanded linear density relationship



References:

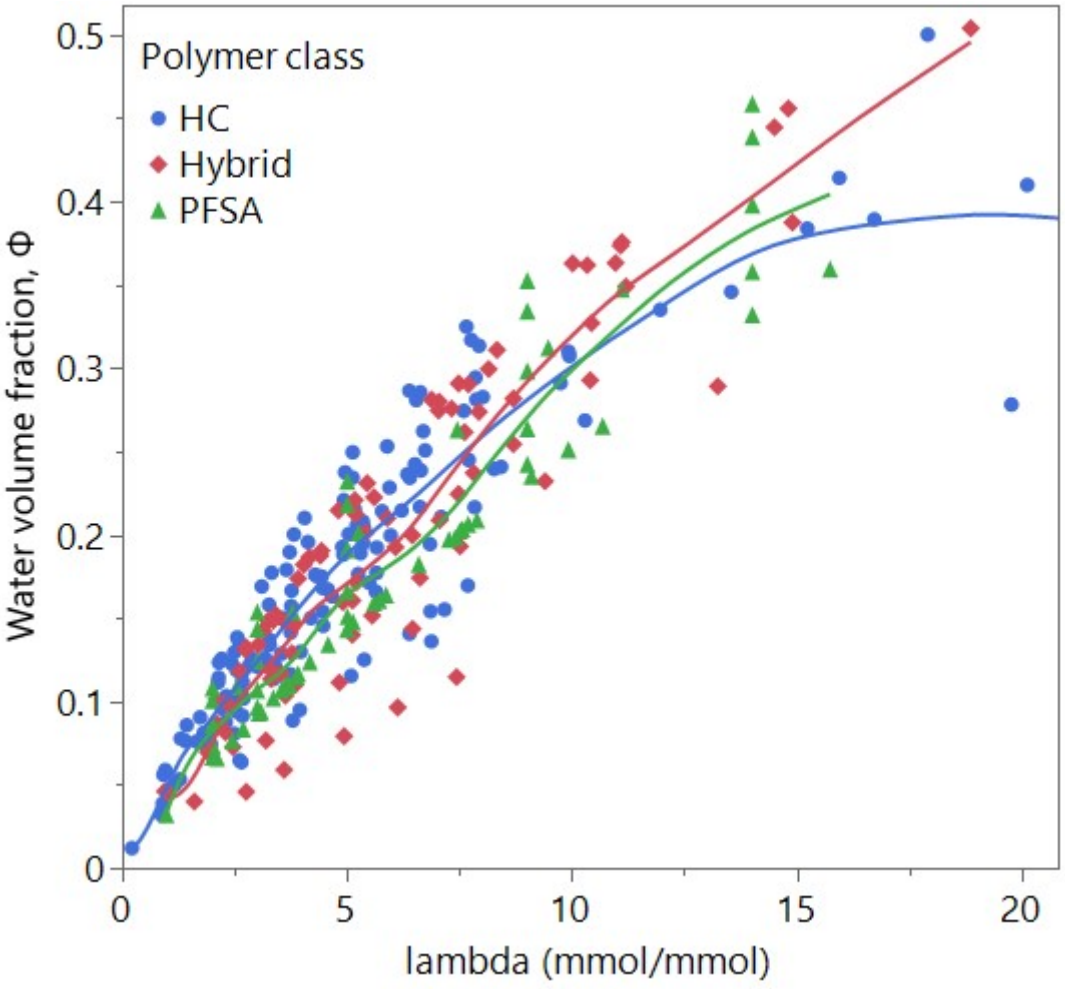
Bae2010	B. Bae, K. Miyatake and M. Watanabe, *Macromolecules*, 2010, **43**, 2684–2691.
Chen2013	S. Chen, R. Hara, K. Chen, X. Zhang, N. Endo, M. Higa, K. Okamoto and L. Wang, *J. Mater. Chem. A*, 2013, **1**, 8178.
Kim2006	Y. S. Kim, B. Einsla, M. Sankir, W. Harrison and B. S. Pivovar, *Polymer*, 2006, **47**, 4026–4035.
Li2011	N. Li, D. S. Hwang, S. Y. Lee, Y.-L. Liu, Y. M. Lee and M. D. Guiver, *Macromolecules*, 2011, **44**, 4901–4910.
Li2012	N. Li, S. Y. Lee, Y.-L. Liu, Y. M. Lee and M. D. Guiver, *Energy Environ. Sci.*, 2012, **5**, 5346–5355.
Roy2008	A. Roy, H.-S. Lee and J. E. McGrath, *Polymer*, 2008, **49**, 5037–5044.

Fig. S5: Water fraction vs RH, no trimming



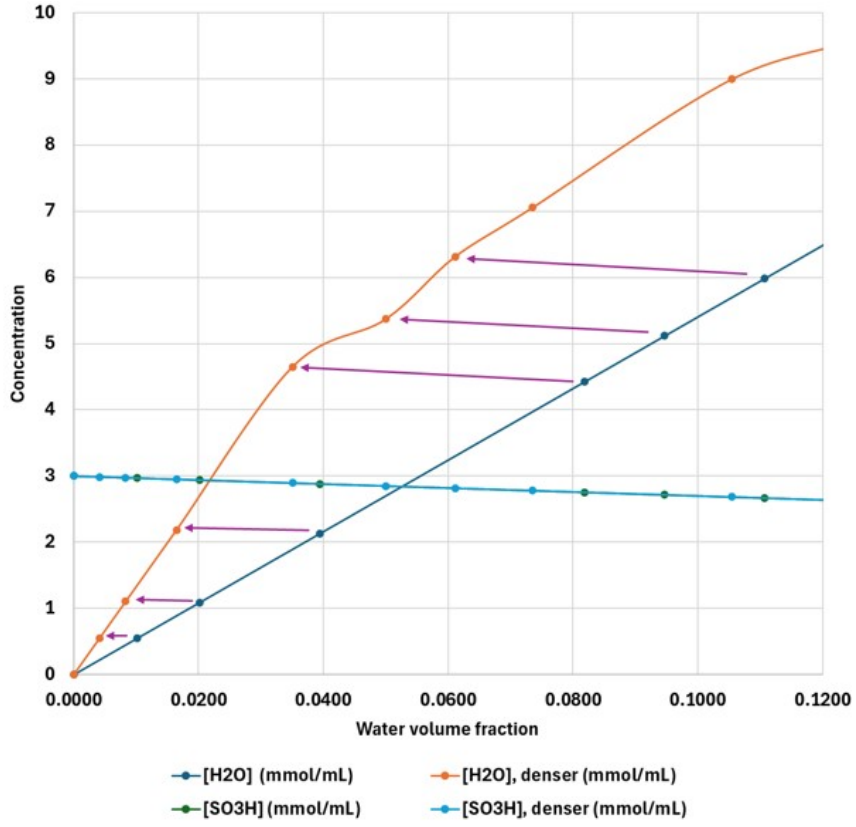
Data taken from references 5,7–31.

Fig. S6: Water volume fraction against lambda (trimmed dataset)



Data taken from references 5,7–31.

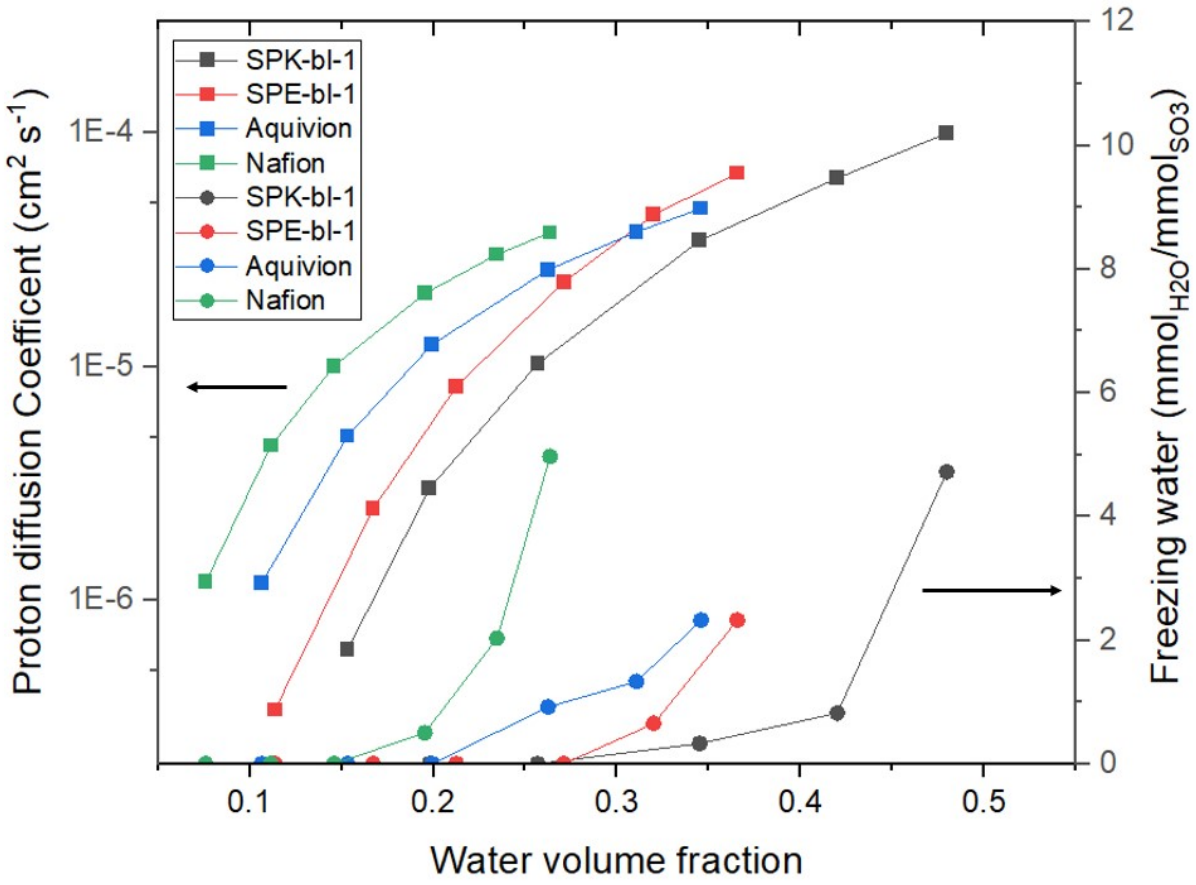
Fig. S7: Impact of water density changes on the water limited regime



Changing water density has a small impact on proton concentration (both in the water and acid). Therefore, the impact on the calculated diffusion coefficient is small.

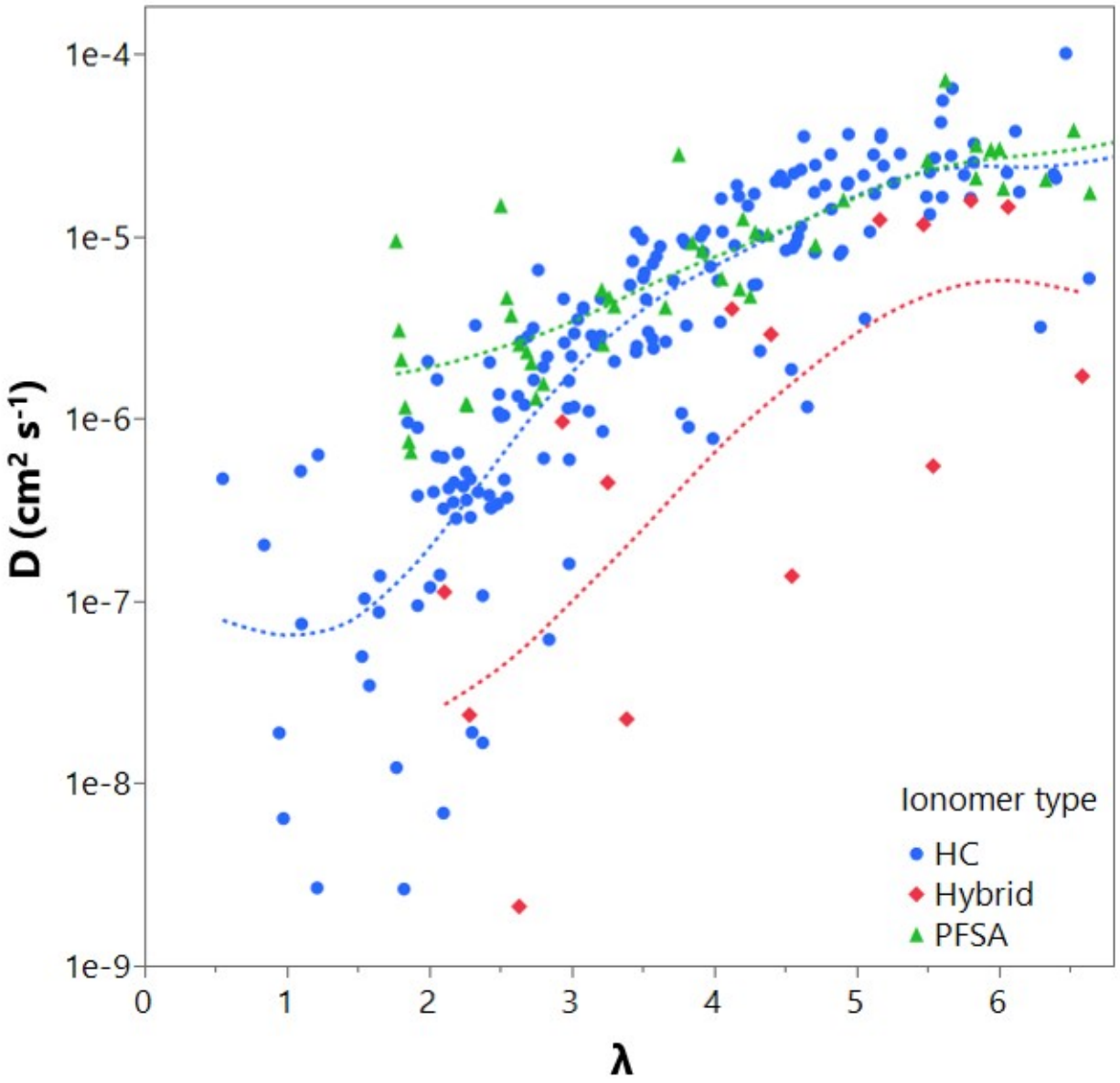
Changing water density has a significant impact on water volume fraction however, increasing density shifts water volume fractions to lower values as an example. This increases the uncertainty of the water volume fractions when at low water fractions (as shown by purple arrows in figure above). Note: Fig. S6 depicts higher density Bai *et al.* showed some polymers with lower densities at low water fractions as well. Data taken from Reference 47.

Fig. S8: Mochizuki bulk and freezing relationship



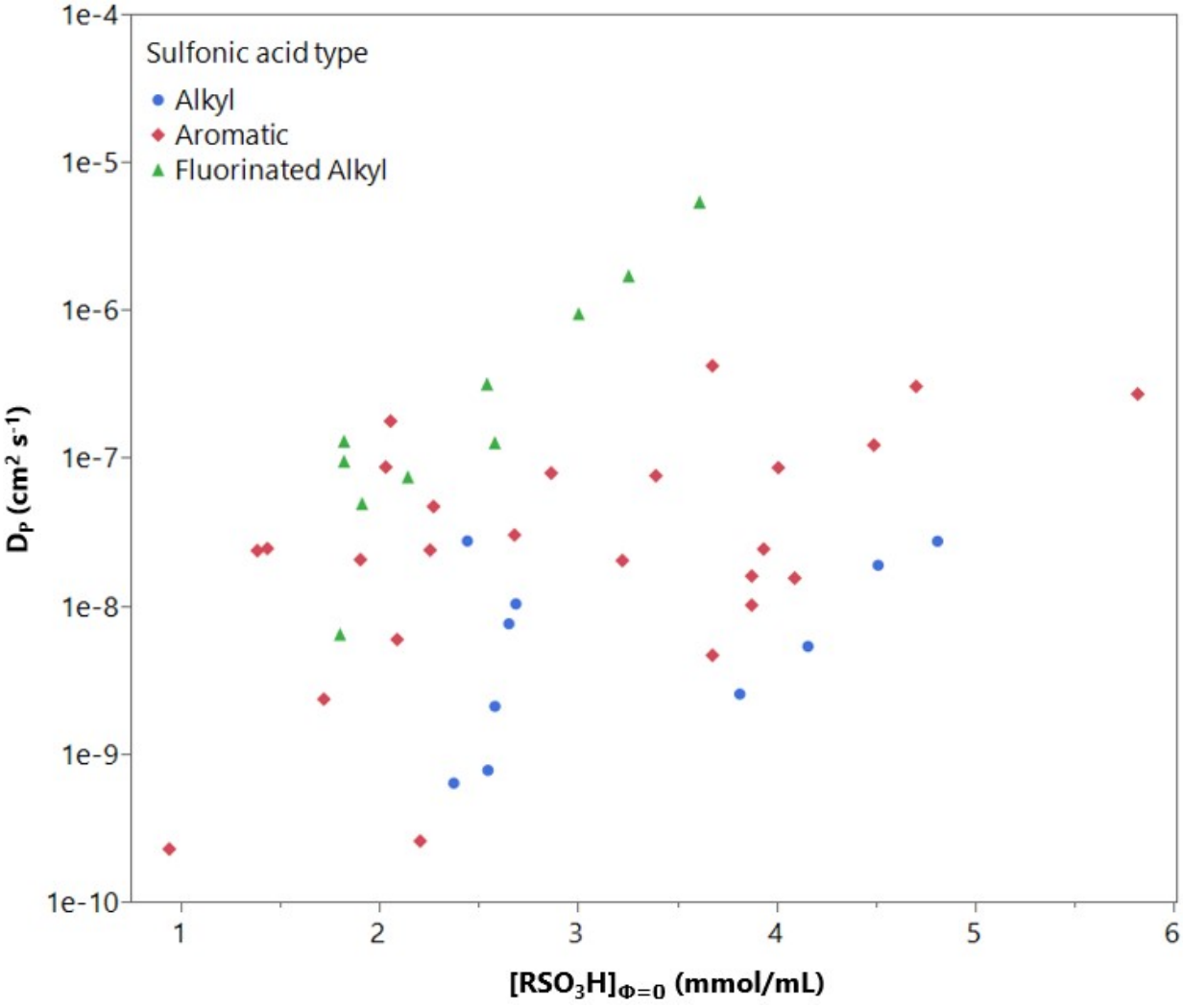
Data from T. Mochizuki, K. Kakinuma, M. Uchida, S. Deki, M. Watanabe and K. Miyatake, ChemSusChem, 2014, 7, 729–733.

Fig. S9: Diffusion coefficients against lambda



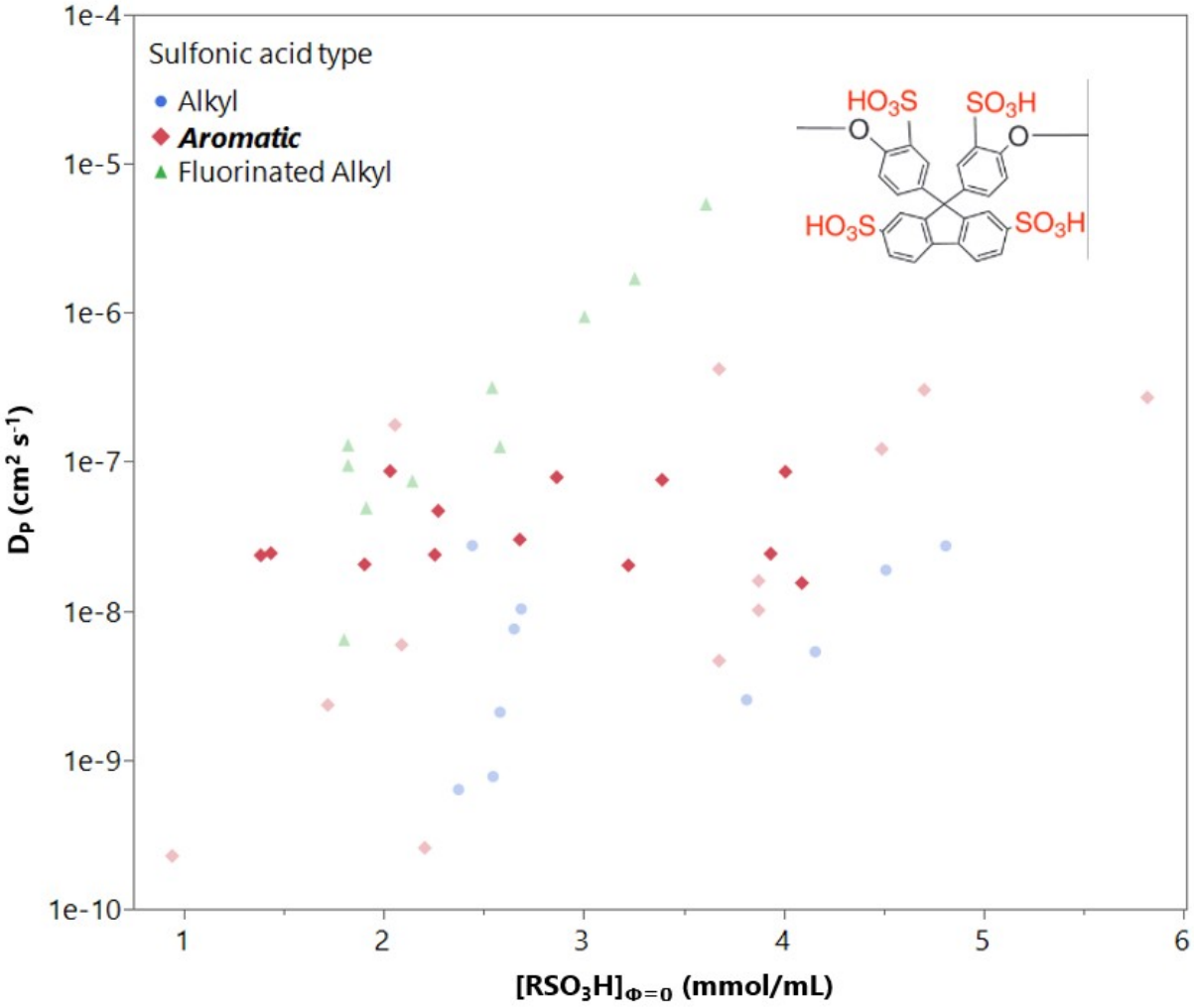
Data from references: 7, 9, 11, 12, 20, 21, 23-25, 28, 31, and 33.

Fig. S10: Dp against sulfonate type



Data taken from references 5,7–31.

Fig. S11: Fluorenylidene biphenylene units highlighted on Dp



Data taken from references 5,7–31.