High Temperature Polymer Electrolyte Fuel Cell Model

Qing Cao

1 Introduction

Many categories of fuel cells (FCs) were developed over the past decade, based on different electrolytes and operating temperature. One promising type of FC is high-temperature polymer electrolyte fuel cell (HT-PEFC), it has a similar structure and functionality with the polymer electrolyte fuel cell (PEFC). The phosphoric acid-doped polymer is usually used as the electrolyte. The operating temperature of this device is between 150 and 180 °C. The HT-PEFC shows several advantages compare to PEFC, such as the relatively high carbon monoxide (CO) tolerance (1–2%) and non-flooding-water in gas channels and gas diffusion layers (GDL). A typical application of HT-PEFC is the fuel-cell-based auxiliary power units (APU) system. The high CO tolerance of HT-PEFC allows people to use reforming gas as the power source and gain a high net system efficiency of the APUs in the order of 20%, which is 2–12% higher than the conventional APUs.

The goal of HT-PEFC development is to develop the system with high performance, high durability and low cost. Therefore, a set of robust models on the cell level and system level are needed, to improve the understanding of the physics, and perform the prediction and optimization of HT-PEFC. The computational fluid dynamics (CFD) is a powerful technology for the modelling of the HT-PEFC. Many CFD modelling works are published based on commercial software such as STAR-CD, FLUENT, PHOENICS and COMSOL. On the other hand, the open-source-CFD

The original version of this chapter was revised: The ESM material has been updated. The correction to this chapter is available at https://doi.org/10.1007/978-3-030-92178-1_10

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/978-3-030-92178-1_2.

Q. Cao (✉)
Forschungszentrum Jülich, Jülich, Germany
e-mail: caoqing85@hotmail.com

S. Beale and W. Lehnert (eds.), *Electrochemical Cell Calculations with OpenFOAM*,
Lecture Notes in Energy 42, https://doi.org/10.1007/978-3-030-92178-1_2

model (based on OpenFOAM) for FCs was developed and published by Beale et al. (2016; http://openfuelcell.sourceforge.net/project).

In this chapter you will learn how to use OpenFOAM to achieve the modeling of HT-PEFC. We will modify the OpenFuelCell library from Beale et al. to create the new model for HT-PEFC. As shown in the last chapter, the OpenFuelCell library solves the heat- and mass transfer equations in channels and porous zones for SOFC. It also provides useful classes, like calculation of multi-component diffusion coefficients, or heat capacity and dynamic viscosity of species. Those design patterns and classes can be further implemented in the HT-PEFC model. However, the major differences between HT-PFEC and SOFC should be noticed and treated carefully in the HT-PEFC model, they are:

- The composition of gas mixture: not like SOFC, the oxidant reduction reaction (ORR) of HT-PEFC happens at the cathode. It means the source term of water should be placed at the cathode catalyst layer. Typically, there are three species in the cathode side: H_2O (gas), O_2 and N_2.
- The electrochemical kinetics: the activation loss and related parameters like the symmetry factor of the energy barrier and the exchange current density are different, based on the characteristics of the membrane, catalyst layer and operating conditions.
- The transfer of water: the membrane and electrodes of HT-PEFC can be considered as flooded with phosphoric acid. The production of water leads to concentration gradients, this in turn causes water diffusion in the acid. Additionally, the water content of membrane changes due to evaporation at the anode and cathode. The overall influence leads to a water transfer from one side to the other, a change in protonic resistance and a swelling of the membrane.

This chapter is organized as follow: first, the working principle, the components and working principle of HT-PEFC are presented. And then the OpenFOAM-model of HT-PEFC is introduced in two sections. In the first section, a basic model is introduced with C++ code, which incorporates the heat- and mass transfer equations and surface-related electrochemistry equations. The simulation results are compared with an analytical model: the Kulikovsky's model (Kulikovsky 2004). In the second section, an extended model of HT-PEFC is performed, to simulate the water crossover in the phosphoric acid flooded polymer membrane. The simulations are validated with the experimental results in Jülich and other publications.

2 Components and Principle of the HT-PEFC

The core part of HT-PEFC consists of Gas diffusion layers (GDLs), bipolar plates (BPPs) and a membrane electrode assembly (MEA). These components are shown in Fig. 1.

The HT-PEFC MEA is very different from other fuel cells: it has a poly(2,2′-mphenylene-5,5′-bibenzimidazole) (PBI) membrane doped with the concentrated

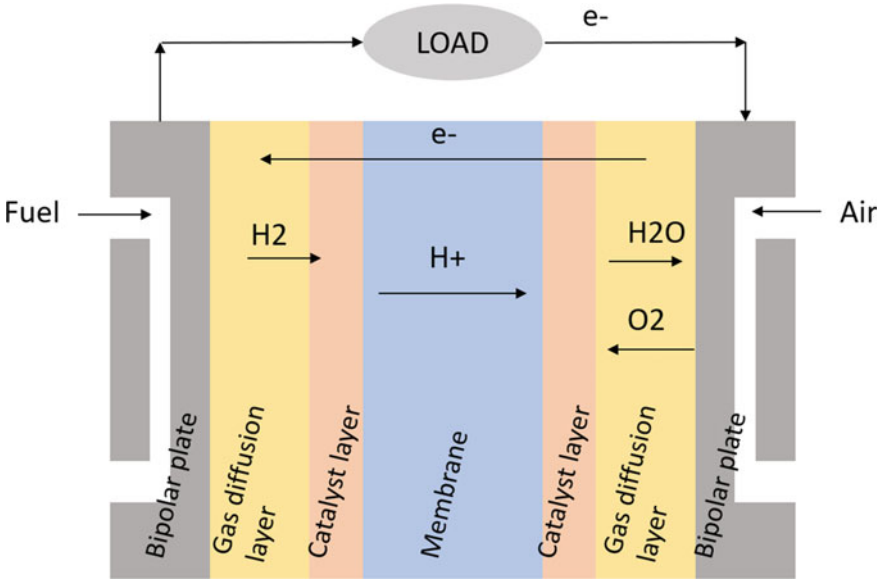


Fig. 1 The components of the HT-PEFC

phosphoric acid and two catalyst layers (CLs) on anode and cathode side (Wainright et al. 1995). The acid doped membrane transfers protons from the anode to the cathode and prevent the passage of gas. That membrane has very good thermal stability, chemical stability and mechanical strength. However, the proton conductivity of the MEA is not constant: it depends on the weight percent (wt%) of phosphoric acid. Previous studies show that by 85 wt% of acid the conductivity reaches maximum. In the CL, the catalyst (Platinum) is mixed with supports (carbon blacks) and binders (polytetrafluoroethylene). The acid infiltrates into CL and the electrochemical reaction happens on the triple phase boundary of the mixture. The GDL has a porous solid structure, for allowing the reactant to move to the CL and the generated water to move to the gas channel. Typically, the GDL consists with carbon fiber paper with total thickness from 100 to 300 μm. The BPP provides the pathway for electrons and contains channels for allowing gas to pass through the flow field. The BPP is usually made by graphite or metal.

When the HT-PEFC starts working, the hydrogen and oxygen in the anode- and cathode side gas channels move through the GDLs and reacts in the CLs. The electrochemical reactions are:

Anode:

$$H_2 \rightarrow 2H^+ + 2e^- \tag{1}$$

Cathode:

$$\frac{1}{2}O_2 + 2H^+ + 2e^- \rightarrow H_2O(g) \quad (2)$$

Overall:

$$H_2 + \frac{1}{2}O_2 \rightarrow H_2O(g) \quad (3)$$

The oxygen reduction reaction (ORR) in cathode is much slower than the hydrogen oxidation reaction (HOR) in anode. Therefore, the cell performance of HT-PEFC is limited significantly by the ORR kinetics.

The cell open-circuit voltage, also the well-known Nernst voltage, is calculated as:

$$E_{Nernst} = \frac{-\Delta G^T}{2F} - \frac{RT}{2F}\ln\left(\frac{p_{H_2O}}{p_{H_2}p_{O_2}^{0.5}}\right) \quad (4)$$

where.

E_{Nernst} = Nernst potential (V).

ΔG^T = Gibbs free energy change at the operating temperature (kJ mol^{-1}).

p_{H2O}, p_{H2}, p_{O2} = normalized partial pressure (p_i/p_0) of H_2O, H_2 and O_2 (-).

In practice, the polarization curve is used to represent the cell performance. It is the plot of cell voltage against current density, as shown in Fig. 2. The cell voltage might be calculated as the difference between the Nernst voltage and the sum of other voltage losses.

$$E_{cell} = E_{Nernst} - \eta_{act} - \eta_{ohm} - \eta_{conc} \quad (5)$$

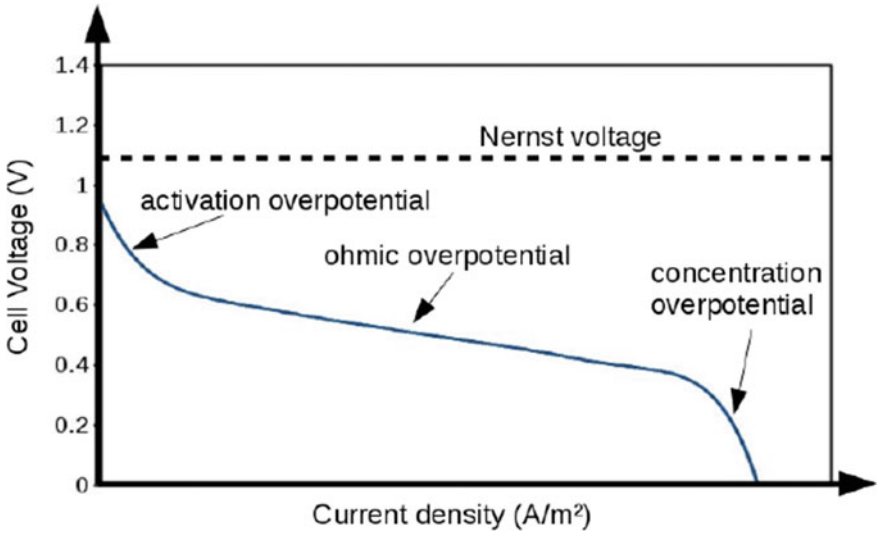


Fig. 2 The polarization curve and all losses of HT-PEFC

where.
E_{cell} = cell voltage (V).
η_{act} = activation overpotential (activation loss) (V).
η_{ohm} = ohmic overpotential (ohmic loss) (V) .
η_{conc} = concentration overpotential (concentration loss) (V).

3 A Basic HT-PEFC OpenFOAM Model

In this section, we are going to create a simple HT-PEFC simulation application with OpenFOAM. The construction of this application contains two parts: create the mesh and create the solver. Let's talk about the mesh first.

3.1 Computational Domain and Meshing

Geometry

The cross-section of the simulated physical domains of a HT-PEFC channel pair are shown in Fig. 4. Different governing equations are solved in different domains: the heat transfer is solved in 'Main domain'; the mass- and momentum transfer are solved in 'Domain I'; the chemical reactions are solved on the interface of 'Domain I' and 'Domain II'. The dimensions h1, h2, … h5 are necessary parameters for creating the mesh, which are shown in Table 1. The typical thickness of CL is 100 μm which is relative thin compare to other components. We may assume 2-D CL surface and ignore the details of the triple phase boundary in this chapter, for reducing the complexity of meshing. Besides the channel pair, people have to notice that in practice the HT-PEFC system includes more hardware such as coolant supplies, current collectors and gaskets. They are not considered in this OpenFOAM model (Fig. 3).

Mesh

The next step is to generate the mesh. For simple flow fields, there are two options for generating meshes. One is to generate a 3D CAD model first, and then generate

Table 1 2-D drawing and parameters of HT-PFC cell structure

Parameter	Symbol	Value	Unit
Height of gas channel	h1	1×10^{-3}	m
Width of gas channel	h2	1×10^{-3}	m
Thickness(total) of bipolar plate	h3	4×10^{-3}	m
Thickness of GDL	h4	3×10^{-4}	m
Thickness of membrane + CLs	h5	2×10^{-4}	m

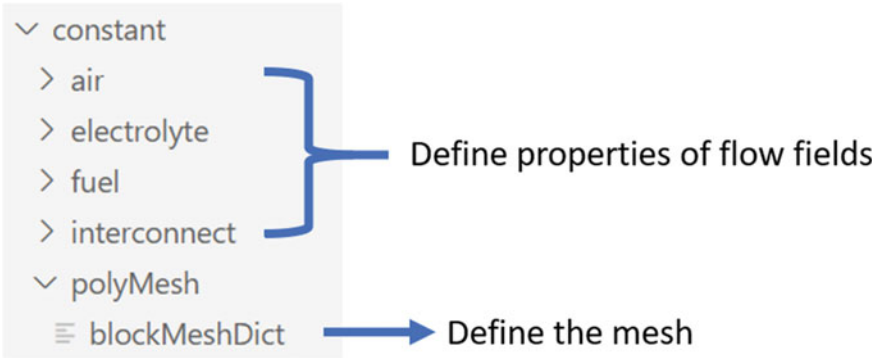


Fig. 3 Code structure of the 'constant' dictionary

a mesh based on the CAD model. The other is to directly generate the mesh with the coordinate matrix using the OpenFOAM mesh-generation utility 'blockMesh'.

There are many options for the toolchain of the first path, such as SALOME Geometry module + SALOME Mesh module, ANSYS SpaceClaim + ANSYS ICEM CFD, CATIA V5 + PointWise, etc.

In this section we use the second path, the 'blockMesh'. First, we have to prepare a dictionary file named *blockMeshDict.* And put it in *constant/polyMesh* directory. The introduction of blockMesh can be found here (https://www.openfoam.com/documentation/user-guide/blockMesh.php).

The code structure of the 'constant' dictionary is shown as follow:

The mesh code in the *blockMeshDict* for a HT-PEFC single channel pair is showed as follow:

```
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
convertToMeters 0.001;
vertices
(
// Interconnect0
   (0 0 z0)
   (length 0 z0)
   (length width z0)
   (0 width z0)
// Interconnect0_to_Air
   (0 0 z1)
   (length 0 z1)
   (length width z1)
   (0 width z1)
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
```

As shown, we create a point coordinate list. In this list we define four points for a surface and eight points for a block. With this rule, the blocks for gas channels, BPPs, GDLs and the membrane of the HT-PEFC single-channel pair can be easily defined. In the remain part of the *blockMeshDict* file, the boundaries are defined by the index of the points:

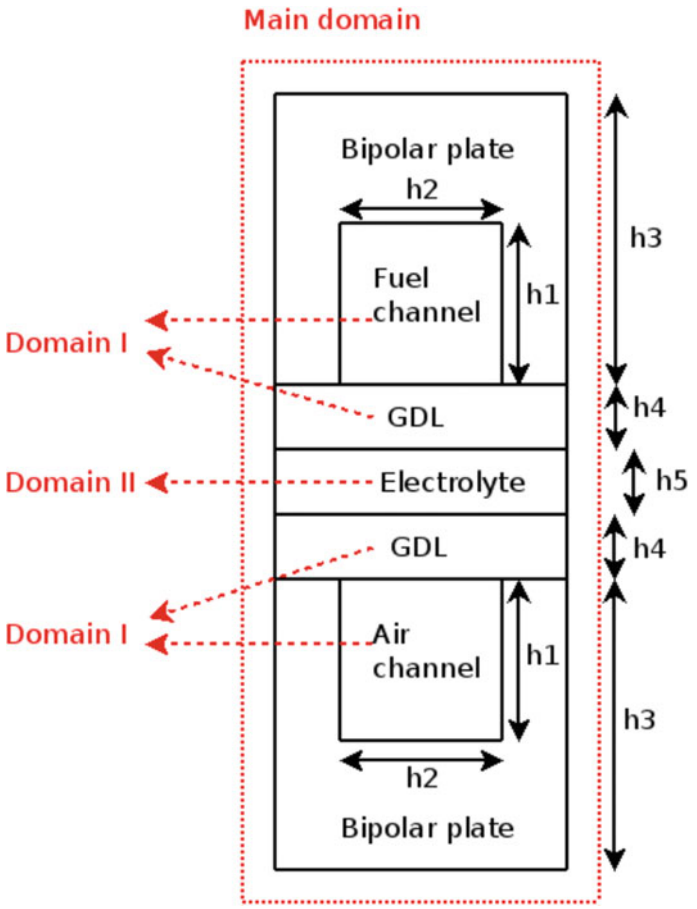


Fig. 4 Cross-section of a single channel pair

```
// ************************************************************ //
  Patch airInlet
    (
       (4 8 11 7)
)
  Patch airOutlet
    (
       (5 6 10 9)
    )
…
// ************************************************************ //
```

After the mesh dictionary prepared, we can generate the mesh and boundaries with the command:

blockMesh -case < case >

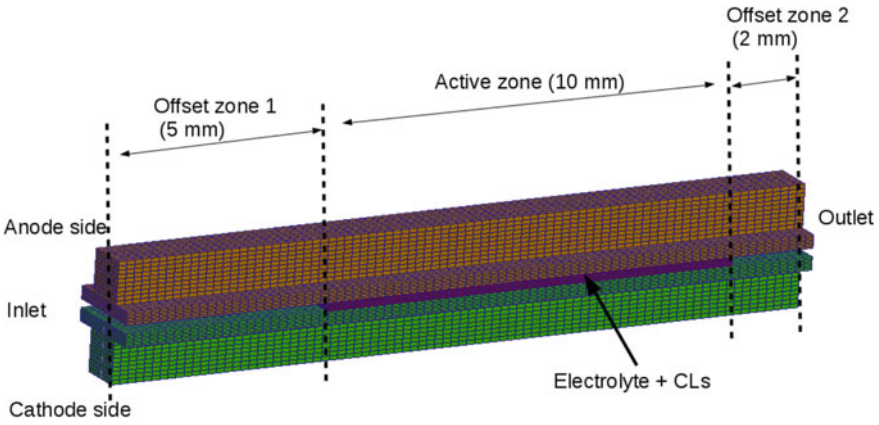


Fig. 5 Mesh of the 17 mm straight single channel (BPP not shown). Brown mesh: anode side GDL and channel; green mesh: cathode side GDL and channel; red mesh: CLs and electrolyte

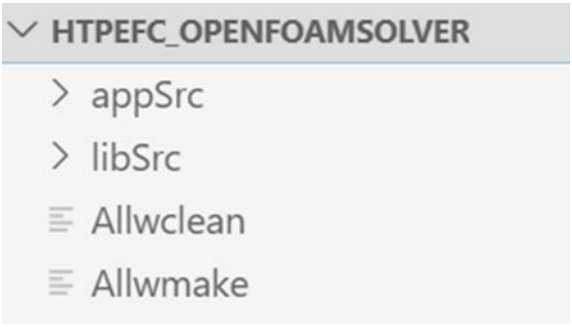
The mesh of the cell (without BPPs) is illustrated in Fig. 5, which is composed of 94,400 hexahedral blocks. Note that two offset zones are included in the mesh, to present the non-reactive gas channel zone of the inlet and outlet.

3.2 Solver Design

Now we have prepared the mesh, let's take a look at the solver. The first step is to design a code structure for this solver. We can use the code structure of the OpenFuelCell project (http://openfuelcell.sourceforge.net/project), as shown below (Fig. 6):

As shown, the project consists of two folders, *appSrc* and *libSrc*. *appSrc* stores code related to the specific fuel cell application, and *libSrc* stores code of general physical models. Readers can download the OpenFuelCell project to check the code.

Fig. 6 HTPEFC Solver structure



To create an HT-PEFC solver based on the OpenFuelCell project, people may do the following:

1. Add new model groups in *libSrc*, such as the electrochemical model group *'catalystLayerModels'*. That model group consists of one parent class and multiple child classes. For the simple HT-PEFC model, we create the electrochemical model file *'simpleNernst.C'*. The code structure is shown as follow (Fig. 7):
2. Modify all *'read_XXX_Properties.H'* files in *appSrc* to read the input of physical properties.
3. Modify all *'create_XXX_Fields.H'* files in *appSrc* to create and initialize the fields used by the solver.
4. Add *'createCatalystLayerModels.H'* in *appSrc* to create an instance of the electrochemical model.
5. Add *'htpefcFoam.C'* to *appSrc*, load all header files in a certain order, and complete the application.

The changes in the appSrc folder are shown in Fig. 8:

All fields should be declared as "create_XXX_Fields" and included by the main routine. A typical declaration looks like this:

```
// ********************************************************* //
   volScalarField Tair
   (
      IOobject
      (
         "T",
         runTime.timeName(),
         airMesh,
         IOobject::READ_IF_PRESENT,
         IOobject::AUTO_WRITE
      ),
```

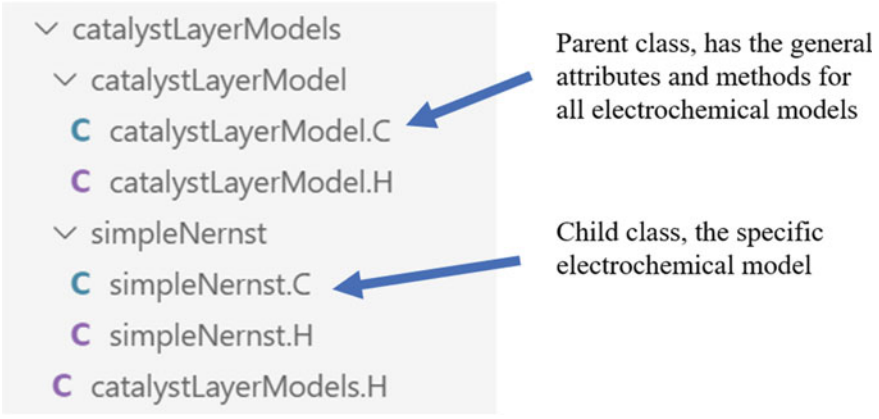


Fig. 7 Code structure of catalystLayerModels

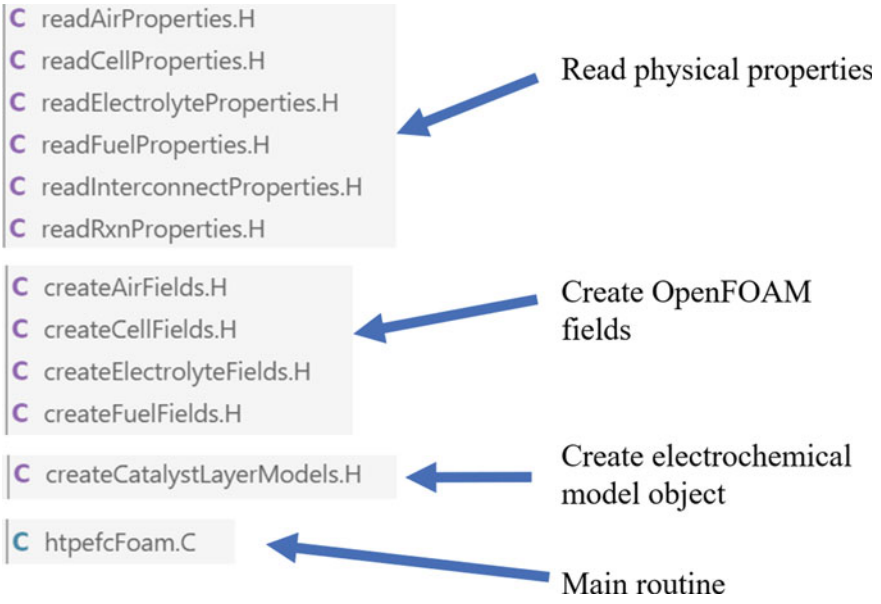


Fig. 8 Changes in appSrc

```
        airMesh,
        Tinit,
        zeroGradientFvPatchScalarField::typeName
    );
// ************************************************************* //
```

In the code above, the `Tair` field is declared and installed on the mesh part `airMesh`. The electrochemical code is shown in the next section with the equations.

3.3 Equations and Parameters

The continuity equation, momentum transfer equation, energy transfer equation and the species transfer equation are same as the SOFC model which introduced in Chap. 1.

It is worth to note that the source term of produced water vapor is not applied in the anode like SOFC, but in the cathode:

$$S_i = \pm \frac{I}{nF} \tag{6}$$

where:
S_i = species source term due to reaction (mol m^{-2} s^{-1}).
I = local current density (A m^{-2}).

F = Faraday constant (C mol^{-1}).
n = electrons number (–).

The species is consumed or produced is defined in *runCase/constant/[field name]/htpefcSpeciesProperties*. For anode side, the code:

```
// ************************************************
H2    H2   2.01594  2  -1  0   130.680;
//    |    |      |  |  |   |
//    |    |      |  |  |   standardEntropy [J/(mol K)]
//    |    |      |  |  | enthalpyOfFormation [J/mol]
//    |    |      |  |  produced=1|inert=0|consumed=-1
//    |    |      |  molecularChargeForFaradaysLaw
//    |    |   molecularWeight [kg/kmol]
//    |    name
// ************************************************************
```

For cathode side, the code:

```
// ************************************************************
O2    O2   31.9988   4   -1  0        205.152;
H2O    H2O 18.01534   2   1   -241.8349e3 188.835;
N2    N2   28.0134   0   0   0        191.609;
// ************************************************************
```

As shown, solver uses '1', '0' or '–1' to define the species is produced, inert or consumed in the cell. Number of electrons in Eq. 6 is also defined as '*molecularChargeForFaradaysLaw*'.

The biggest difference between HT-PEFC model and SOFC model is how to calculate the Nernst voltage and the activation overpotential. The Nernst voltage is given by:

$$E_{Nernst} = E_{th}^{T} - \frac{RT}{2F} \ln\left(\frac{X_{H_2O}}{X_{H_2} X_{O_2}^{0.5}}\right) \tag{7}$$

where:
E_{Nernst} = Nernst potential (V).
E_{th}^{T} = thermodynamic voltage at the cell operating temperature (V).
X_{H2O}, X_{H2}, X_{O2} = mole fraction of species H_2O, H_2 and O_2 (–).
The code for Nernst voltage is in '*solver\appSrc\NernstEqn.H*':

```
// ************************************************************
Nernst =
relaxNernst.value()*(E0.value() - Rgas*anodeT*(Foam::log(XH2O)-
Foam::log(XH2)-0.5*Foam::log(XO2))/(2*F))          +          (1-
relaxNernst.value()) * Nernst;
// ************************************************************
```

Note that the `XH2`, `XH2O` and `XO2` in above code are field value, the calculated Nernst voltage is also a field. The under-relaxation is applied to stabilize the calculation by limiting the rate of change of fields.

The activation overpotential is given by:

$$\eta_{act} = \frac{RT}{\alpha F} \ln\left(\frac{I}{i_0} \frac{X_{ref}}{X_{O_2}}\right) \tag{8}$$

where:
i_0 = exchange current density for the ORR (A m^{-2}).
α = symmetry factor of the reaction (–).
X_{ref} = reference mole fraction (–).
X_{O2} = oxygen mole fraction (–).
The exchange current density is given by:

$$i_0 = i_0^{ref} \exp\left(-\frac{E_A}{R}\left(\frac{1}{T} - \frac{1}{T_{ref}}\right)\right) \tag{9}$$

where:
$i_0{}^{ref}$ = reference exchange current density for the ORR (A m^{-2}).
E_A = activation energy for the ORR (J mol^{-1}).
T_{ref} = reference temperature of operating (K).
The local current distribution is calculated by:

$$I = \frac{E_{Nernst} - E_{cell} - \eta_{act}}{R_\Omega} \tag{10}$$

where:
E_{Nernst} = Nernst potential (V).
η_{act} = activation loss (V).
I = current density of local electrolyte (A m^{-2}).
R_Ω = ohmic resistance (specific) (Ω m^{-2}).
The calculation of activation overpotential (field), exchange current density (single value) and current (field) are coded in *'solver\libSrc\catalystLayerModels\simpleNernst.C':*

```
// ************************************************************
#include "simpleNernst.H"
namespace Foam
{
   namespace catalystLayerModels
   {
      defineTypeNameAndDebug(simpleNernst, 0);
      // Constructors
      simpleNernst::simpleNernst
      (
         scalarField& XO2,
         scalarField& YO2,
         scalarField& i,
         scalarField& eta,
         scalarField& Nernst,
         const dictionary& dict,
         dimensionedScalar& Temp,
         dimensionedScalar& Vcell
      )
      :
```

```
        catalystLayerModel(XO2, YO2, i, eta, Nernst, dict),
        j0Ref_(dict_.lookup("j0Ref")),
        Tref_(dict_.lookup("Tref")),
        alpha_(dict_.lookup("alpha")),
        Rohm_(dict_.lookup("Rohm")),
        F_(dict_.lookup("F")),
        Rgas_(dict_.lookup("Rgas")),
        EA_(dict_.lookup("EA")),
        relax_(dict_.lookup("relax")),
        XO2ref_(dict_.lookup("XO2ref")),
        Temp_(Temp),
        Vcell_(Vcell)
        {}
        // Members functions
        void simpleNernst::evaluate()
        {
        scalar kt = (Rgas_.value() * Temp_.value()) / (F_.value() * \\
alpha_.value());
scalar j0 = j0Ref_.value()* Foam::exp( (-EA_.value()
/ Rgas_.value()) * (1.0 / Temp_.value() - 1.0 / Tref_.value() ) );
          eta_ = kt * Foam::log(i_/(j0 * (XO2_/XO2ref_.value())));
          i_  = relax_.value()*(Nernst_ - Vcell_.value() - eta_)/
 Rohm_.value() + (1-relax_.value())* i_;
        }
    }
}
// ************************************************************
```

Equations 8–10 are solved in the members function `simpleNernst::evaluate()`. The under-relaxation is used for calculating the current. Note that in the constructor, the necessary `scalarField` such as `XO2, i` or `Nernst` has to be declared and pointed to an initialized `scalarField` with pointer symbol `&`. The physical constants are read with the `dict_.lookup()` function. The `simpleNernst` model object itself is created with *solver\appSrc\ createCatalystLayerModels.H:*

```
// ************************************************************
…
          const dictionary& simpleNernstDict = cellProper-
ties.subDict("simpleNernst");
   catalystLayerModels::simpleNernst simpleNernst = catalystLay-
erModels::simpleNernst(XO2, YO2, j_O2CL, eta, Nernst, simpleNern-
stDict, Temperatur, V);
…
// ************************************************************
```

The heat capacity and dynamic viscosity of species by typical operating temperature of HT-PEFC are calculated using 6th order polynomial according to Todd and Young's research. The other modelling parameters are shown in Table 2. Those parameters are assumed to be constant and easily to be implemented to the code. By default, OpenFOAM uses the SI units with 7 scalars delimited by square brackets. The parameters are defined in '*runCase/constant/cellProperties*':

Table 2 Modelling parameters of the basic model

Parameter	Symbol	Value	Unit
Symmetry factor	α	0.5	
Ohmic resistance of cell per m^2	R_Ω	2×10^{-5}	Ωm^{-1}
Reference exchange current density	i_0^{ref}	3.0	Am^{-2}
Reference mole fraction of oxygen	X_{ref}	0.23	
Reference temperature	T_{ref}	433	K
Thermodynamic voltage	E_{th}^T	1.15	V
Activation energy	E_A	57100	$Jmol^{-1}$
Thickness of membrane + CLs	l_e	2×10^{-4}	m

```
// ************************************************************
…
alphaalpha[0 0 0 0 0 0 0] 0.5;
Rohm Rohm [1 4 -300 -20] 2.0e-5;
…
// ************************************************************
```

Note that by other cell assembly the parameters should be different, due to the different porous GDL, composite of catalyst layer, situation of membrane, etc.

3.4 Results

Polarisation curve

In this chapter, eight operating conditions with average of the local current densities from 1000 A/m^2 to 8000 A/m^2 were used to calculate the operating range of an HT-PEFC.

The operating temperature was 433 K, the gas pressure at both anode and cathode side are 101325 Pa, the mass fraction of species is: 100% H_2 at anode, 23% O_2 and 77% N_2 at cathode. Stoichiometric ratio at both sides is 2.

For each operating current, the cell voltage was calculated by the solver according to Eq. 10. A polarization curve was therefore plotted, as shown in Fig. 9. People can find that the polarization curve of CFD-simulations fits the in-house experiments well.

Local distribution

Since the cathode side species distribution determines the performance of the HT-PFEC, we will use cathode side in this chapter to show the numerical results. In the 3D flow field of the cathode, we choose two section-plane to show the distribution of the species: plane A and B. In Fig. 10. we put the plane A in the center of the channel and put plane B on the electrolyte-GDE interface.

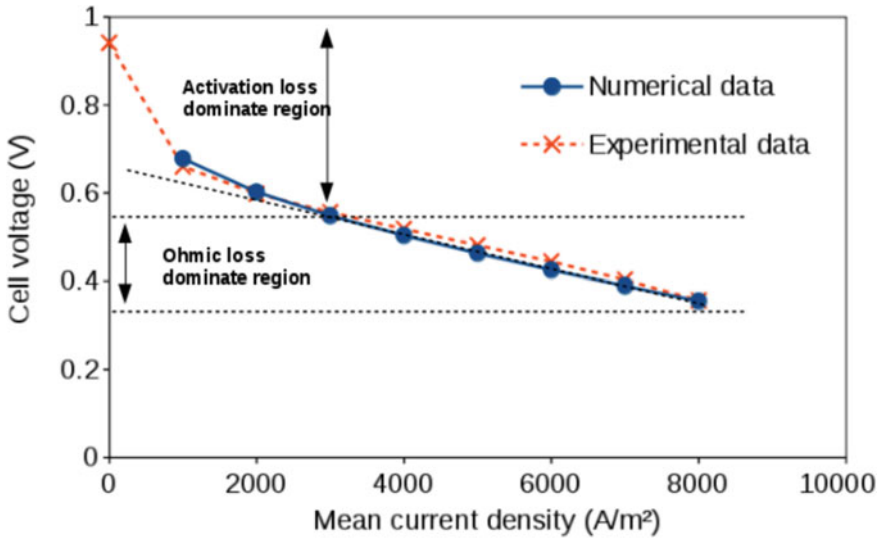


Fig. 9 Comparison of numerical and the experimental results of the polarisation curve

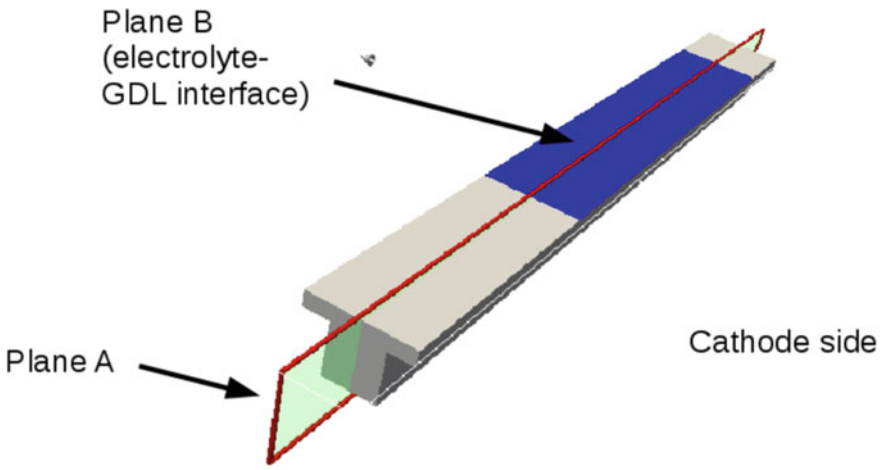


Fig. 10 The position of the cross-sectional plane A and B. To present the local simulation results

We may choose 2 operating points to compare the species distribution at the low- and high current density: 2000 and 8000 A m^{-2}. The oxygen mole fraction on plane A and plane B of these 2 sampling points are shown in Figs. 11 and 12.

For HT-PEFC, there is an analytical model by A. Kulikovsky (2004) to calculate the mole fraction of oxygen in the channel. Note that this equation assumed plug flow.

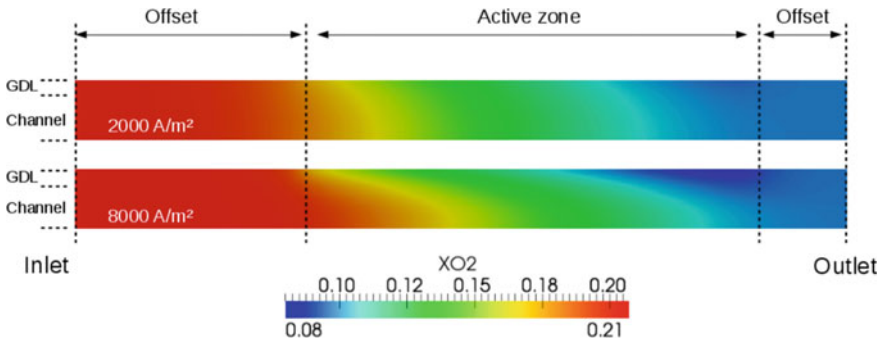


Fig. 11 Local distribution of oxygen mole fraction at 2000 A m^{-2} and 8000 A m^{-2} on Plane A

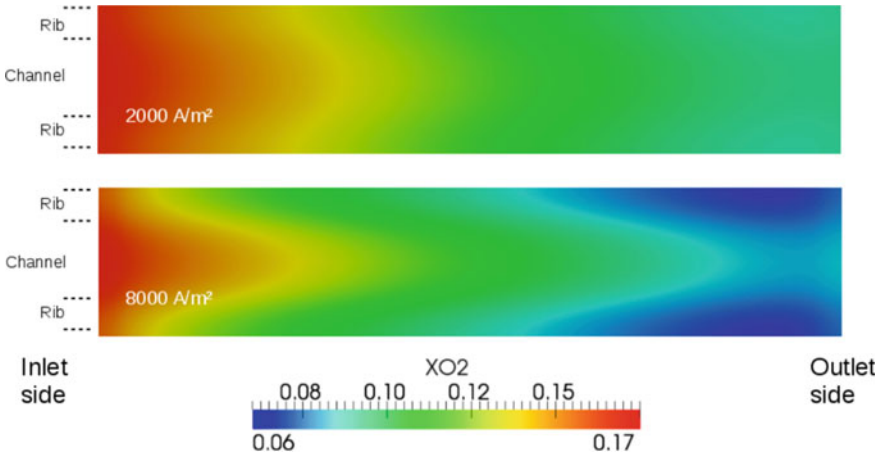


Fig. 12 Local distribution of oxygen mole fraction at 2000 A m^{-2} and 8000 A m^{-2} on Plane B

$$X_{O_2}(x) = X^0_{O_2}\left(1 - \tfrac{1}{\lambda}\right)^{\frac{x}{x^0}} \tag{11}$$

where:
$X_{O2}(x)$ = oxygen mole fractions along the channel direction.
$X_{O2}{}^0$ = oxygen mole fractions at the input.
λ = stoichiometric ratio of HT-PEFC.
x = distance (m).
x_0 = length of the reaction zone (m).

This model can be used to verify our numerical results. Use the same operating parameters as the CFD, we can get the analytical curve of X_{O2}. The difference between numerical and analytical calculation is shown in Fig. 13. People should note that the X_{O2} in the Kulikovsky model is a singel value and in the CFD model is a field value. In Fig. 13 we averaged the CFD result of X_{O2} to compare it with the Kulikovsky model.

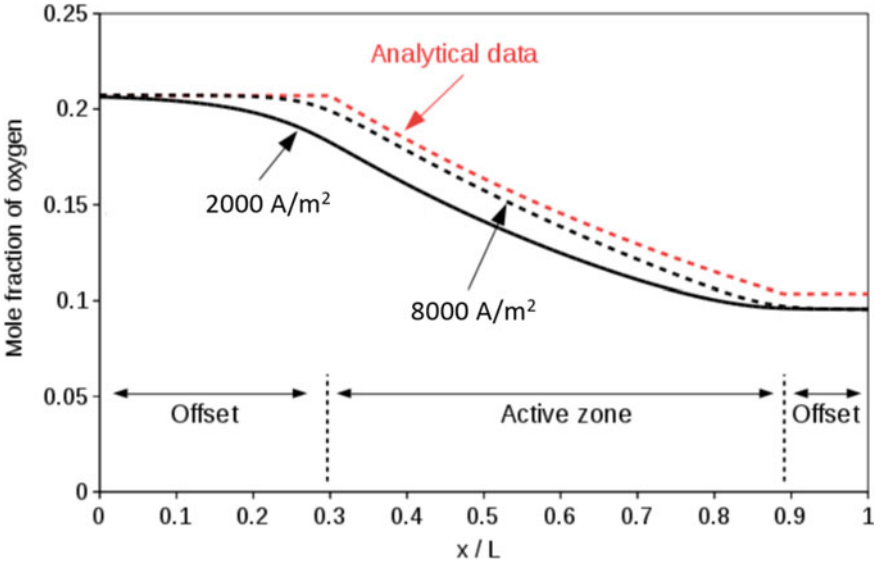


Fig. 13 Comparison of oxygen mole fraction distribution along the channel at 2000 A m^{-2} and 8000 A m^{-2} on Plane A

People can see that the CFD-simulation and the analytical calculation have similar trends: the mole fractions of oxygen decrease along the gas channel direction. However, the CFD result at high current fit better with the analytical results than low current. The reason is that the Péclet number at low current is much smaller than high current, so the diffusion effect becomes obvious.

4 An Extended HT-PEFC OpenFOAM Model

The previous section we describe how to build a simple HT-PEFC modeling project. In this chapter, we will show how to add some advanced modules to the basic model.

In HT-PEFC, the water balance affects the cell performance significantly, because (i) in the membrane, the proton conductivity depends significantly on the phosphoric acid concentration (Wainright et al. 1995) and (ii) the partial pressure of water vapor on electrode affects the kinetics. The Fig. 14. shows the water transport process during cell operation: water is first produced at the membrane-GDE interface of the cathode, and then water vapor is absorbed into the membrane, resulting in a difference in water concentration on both sides of the membrane. The water flows from the cathode side to the anode side due to the chemical potential, and finally enters the GDL from the anode side. The process of water entering and leaving the membrane is controlled by the evaporation-adsorption process.

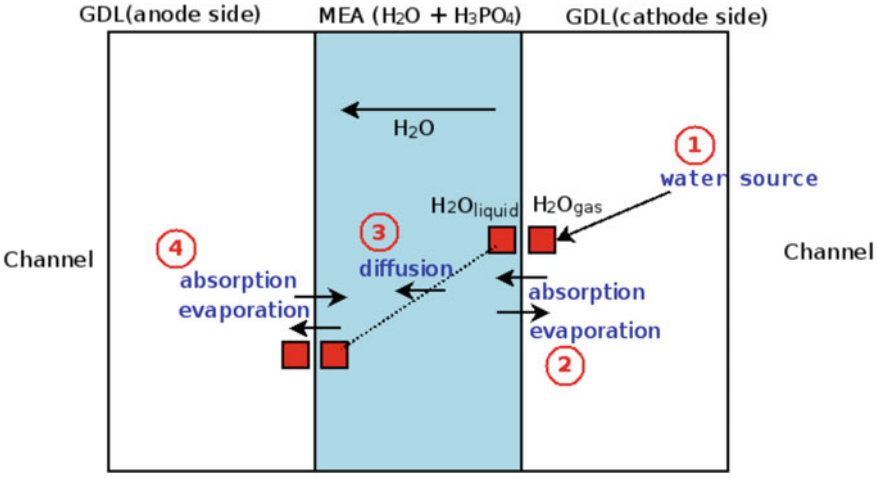


Fig. 14 Schematic model of water transfer in electrolyte

4.1 Computational Domain and Meshing

Geometry

In this section we will create the mesh corresponds to an existing HT-PEFC cell (Fig. 15.) in Jülich. The active area is around 50 cm^2. The dimension of cross-section of the simulated physical domains are the same as last section.

Mesh

We cannot use the 'blockMesh' for this project due to the complexity of the geometry. For such complex shapes, we can use the open-source pre-processing tool ‚Salome'.

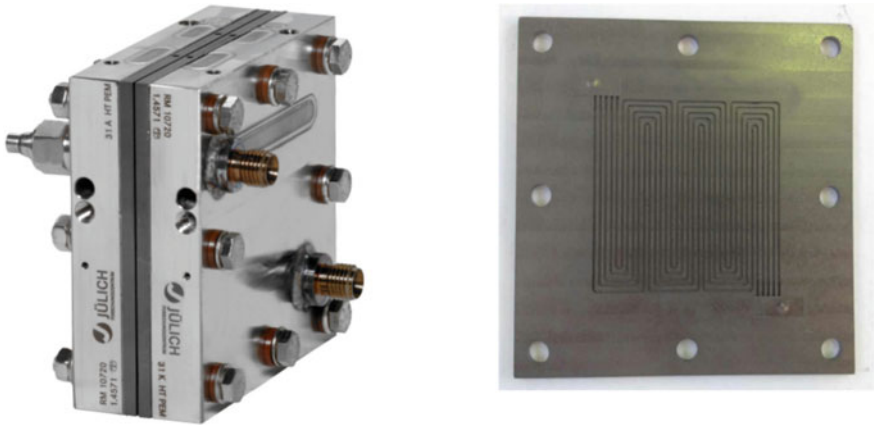


Fig. 15 *Left: 5-channel serpentine test cell in Jülich; right: the bipolar plate of this cell* (Cao 2017)

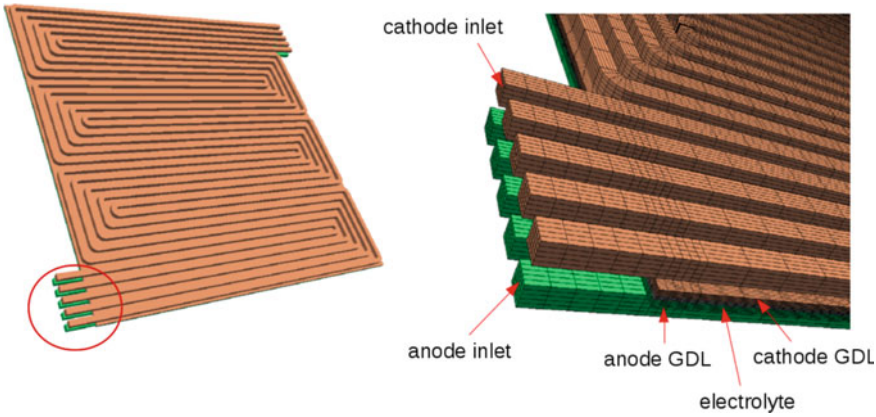


Fig. 16 Left: 3-D geometry of the 5-channel serpentine test cell (BPP not shown); right: the mesh of inlet area

There is truly marvelous meshing process with Salome, which this margin is too narrow to contain. The introduction of Salome Meshing can be found here (https://docs.salome-platform.org/7/gui/SMESH/index.html). The mesh of this single cell is illustrated in Fig. 16, which composed of 0.74 million hexahedral volumes.

4.2 Solver Design

We have already completed the project architecture in the previous chapter. Adding new modules base on that becomes simple:

1. Add water transfer model groups 'waterTransferModels' in libSrc, create parent class 'waterTransferModel' add child class 'diffusionDrive'. The code structure is shown as follow (Fig. 17):
2. Modify all 'read_XXX_Properties.H' files in appSrc to read the input of physical properties of new module.
3. Modify all 'create_XXX_Fields.H' files in appSrc to create and initialize the fields of new module.
4. Add 'createWaterTransferModels.H' in appSrc to create an instance of the water transfer model.
5. Modify 'htpefcFoam.C' in appSrc, load all header files in a certain order, and complete the application.

The specific code will be shown in the next section based on the water transfer equations.

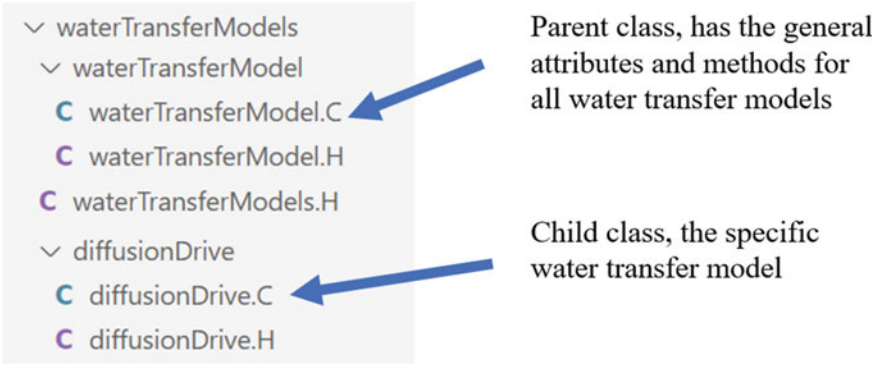


Fig. 17 Code structure of waterTransferModels

4.3 Equations and Parameters

In the H_3PO_4 electrolyte, the diffusion flux of the water a function of the gradient of the liquid water mass fraction. That diffusion flux is given by:

$$j_{H_2O} = \frac{\psi k_{H_2O,l}^{eff}\left(Y_{H_2O}^{liq.c} - Y_{H_2O}^{liq.a}\right)}{l_e} \tag{12}$$

where:
ψ = empirical parameter of the effect of mean current (-).
$Y_{H2O}^{liq.c}$ = liquid water mass fraction at cathode (-).
$Y_{H2O}^{liq.a}$ = liquid water mass fraction at anode (-).
l_e = thickness of electrolyte (m).

Note that we introduce an empirical factor ψ here. Because by different mean current densities, the water-acid solution density also changes. Therefore in the electrolyte, the effective mass transfer coefficient of water might be calculated as a function of current density:

$$\psi = \frac{i_m}{i_{ref,w}} + 0.75 \tag{13}$$

where:
i_m = mean current density (A m^2).
$i_{ref,w}$ = reference current density (A m^2).

The boundary values of liquid water mass fraction are governed by water vapor partial pressure on the gas–liquid interface. Using the research from Schechter et al. (2009), we may get an empirical equation by fitting of their experimental data:

$$Y_{H_2O}^{liq,b} = 0.517\frac{p_{H_2O}}{p_{sat}} + 0.046 \tag{14}$$

where:

$Y_{liq,b}$ = liquid water mass fraction on the gas–liquid interface (-).
p_{H2O} = water vapor partial pressure (Pa).
p_{sat} = saturation water vapor pressure (Pa).

The value of the saturation vapor pressure of water is calculated by Antoine equation (an Clausius-Clapeyron equilibrium formulation):

$$p_{sat} = 133.322 * 10^{a - \frac{b}{c + \tau_0 - 273.15}} \tag{15}$$

where a = 8.14, b = 1810.94, c = 244.49 are empirical factors from an online database: Dortmund Data Bank (http://ddbonline.ddbst.de/AntoineCalculation/AntoineCalculationCGI.exe), tau is the temperature.

Other modelling parameters of this model are listed in Table 3.

Since the saturation vapor pressure of water (Eq. 15) and activity of vapor p_{H2O} / p_{sat} are general fields that can be used for all water transfer models, they should be solved in the parent class *'solver\libSrc\waterTransferModels\waterTransferModel\waterTransferModel.C'*:

```
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
…
      // Member Functions
      //- solve the saturated vapor pressure of water in H3PO4
      void waterTransferModel::solvePsat()
      {
         // solve partial pressure of water
         Info << nl << "Solving partial pressure" << endl;
         pPAir_= pAir_*XH2Ocathode_;
         pPFuel_= pFuel_*XH2Oanode_;
         // solve saturation vapor pressure for pure H2O, DDBST
      Info << nl << "Solving water saturation vapor pressure" << endl;
                              pWSAir_  =  pow(10,  AEP_A_.value()-
AEP_B_.value()/(AEP_C_.value()+Tair_ -273.15) ) * 133.322;
                               pWSFuel_=  pow(10,  AEP_A_.value()-
AEP_B_.value()/(AEP_C_.value()+Tfuel_ -273.15) ) * 133.322;
         // solve water activity
         Info << nl << "Solving water activity" << endl;
         aWAir_ = pPAir_/pWSAir_;
         aWFuel_ = pPFuel_/pWSFuel_;
      }
      void waterTransferModel::relaxGamma()
      {
                        gamma_  =  relax_.value()  *  gamma_  +  (1-
relax_.value()) * gammaOld_;
```

Table 3 Modelling parameters in the water transfer model

Parameter	Symbol	Value	Unit
Effective mass transfer coefficient	$k_{H2O,l}^{eff}$	2×10^{-6}(assumed)	$kgm^{-1}s^{-1}$
Reference current density	$i_{ref,w}$	1×10^{4}(assumed)	Am^{-2}

```
        infoScalarField(gammaOld_, "gammaOld");
    }
…
// ************************************************************
```

Equations 12–14 are solved in the child class *'solver\libSrc\waterTransferModels\diffusionDrive\diffusionDrive.C':*

```
// ************************************************************
#include "diffusionDrive.H"
#include "volFields.H"
namespace Foam
{
  namespace waterTransferModels
  {
    defineTypeNameAndDebug(diffusionDrive, 0);
    // Constructors
    diffusionDrive::diffusionDrive
    (
      scalarField& XH2Oanode,
      scalarField& XH2Ocathode,
      scalarField& gamma,
      const dictionary& dict,
      scalarField& YwaterC,
      scalarField& YwaterA,
      scalarField& prodFluxWater,
      scalarField& pAir,
      scalarField& pFuel,
      scalarField& Tair,
      scalarField& Tfuel,
      scalarField& Telectrolyte
    )
    :
   waterTransferModel(XH2Oanode, XH2Ocathode, gamma, dict, YwaterC,
\\ YwaterA, pAir, pFuel, Tair, Tfuel, Telectrolyte),
    prodFluxWater_(prodFluxWater),
    lMem_(dict_.lookup("lMem")),
    Res_(dict_.lookup("Res")),
    kWater_(dict_.lookup("kWater"))
    {}
    // Member Functions
    void diffusionDrive::evaluate()
    {
      writeData();
      gammaOld_ = gamma_;
      solvePsat();
                                  //  solve  water  mass  frac-
tion boundary in membran, fitting from Schechter et al.
      YwaterC_ = 1-(-51.7 * aWAir_ +95.4)/100;
      YwaterA_ = 1-(-51.7 * aWFuel_ +95.4)/100;
      // solve water flux through membrane
        fluxWater_ = Res_.value()*kWater_.value() * (YwaterC_-
YwaterA_) / lMem_.value();
      // update gammaWater
```

```
            gamma_= fluxWater_ / prodFluxWater_;
            relaxGamma();
        }
    }
}
// ************************************************************
```

4.4 Results

Water outcomes on electrodes

In this section, the water transfer models are validated with the experiment: an in-house HT-PEFC cell was operated with the same conditions as the simulations (Reimer et al. 2016). The water was condensed by two condensers at outlets of anode and cathode and then collected using bottles. Figure 18. shows the comparison of the numerical results and the experiment results. One can see that the water outcomes and the water crossover rate of the CFD simulations fit the experiments well.

Local distribution

The CFD model calculated the scalar field of water and H_3PO_4 in the electrolyte of HT-PEFC. The sum of the mass fractions of water and H_3PO_4 equals 1. Figure 19. shows the mass fractions of H_3PO_4 at the electrolyte-GDE interface. One can see that the H_3PO_4 concentration of the anode is higher than cathode. That concentration

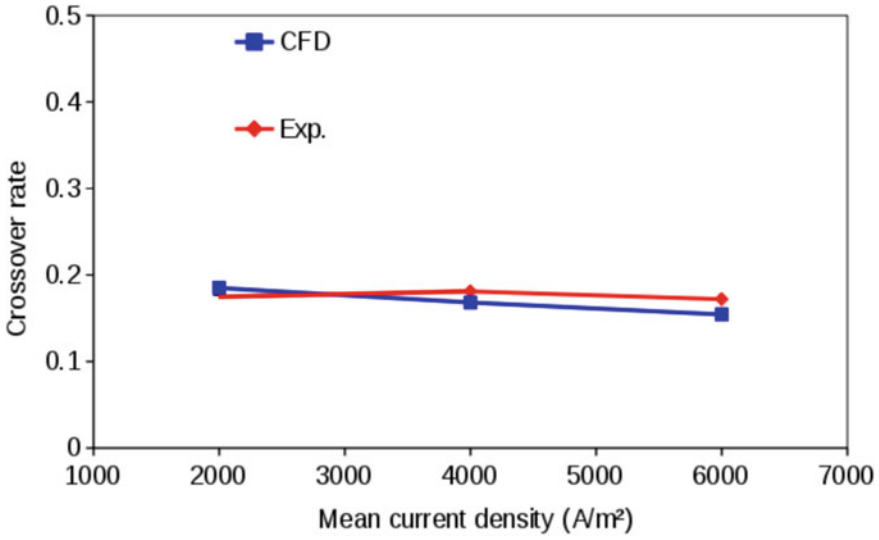


Fig. 18 Cross over rate of the produced water: comparison of CFD and experiment (Cao 2017)

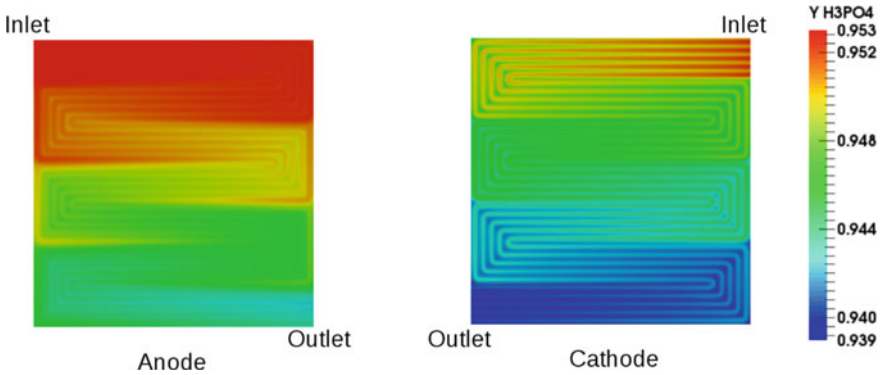


Fig. 19 H_3PO_4 Mass fraction in the electrolyte: comparison of anode and cathode

gradient is the driving force of the water cross over. In the membrane of HT-PEFC, the CFD calculation indicates that mass fractions of H_3PO_4 is between 93.9% and 95.3%. This result is validated with a previous experiment, which applied synchrotron-based X-ray tomographic microscopy to an HT-PEFC single cell: the mass fractions of H_3PO_4 is reported as 96.5 ± 1.5 wt % at 160 °C (Eberhardt et al. 2015).

References

Beale SB, Choi H-W, Pharoah JG, Roth HK, Jasak H, Jeon DH (2016) Open- source computational model of a solid oxide fuel cell. Comput Phys Commun 200:15–26

Cao Q (2017) Modelling of high temperature polymer electrolyte fuel cells. Universitätsbibliothek der RWTH Aachen, Diss

Eberhardt S, Toulec M, Marone F, Stampanoni M, Büchi F, Schmidt T (2015) Dynamic operation of HT-PEFC: in-operando imaging of phosphoric acid profiles and (Re) distribution. J Electrochem Soc 162(3):F310–F316

Kulikovsky A (2004) The effect of stoichiometric ratio on the performance of a polymer electrolyte fuel cell. Electrochim Acta 49(4):617–625

Mesh generation with the blockMesh utility, https://www.openfoam.com/documentation/user-guide/blockMesh.php

Reimer U, Ehlert J, Janssen H, Lehnert W (2016) Water distribution in high temperature polymer electrolyte fuel cells. Int J Hydrogen Energy 41(3):1837–1845

Salome Mesh User's Guide, https://docs.salome-platform.org/7/gui/SMESH/index.html

Saturated Vapor Pressure Calculation by Antoine Equation, http://ddbonline.ddbst.de/AntoineCalculation/AntoineCalculationCGI.exe

Schechter A, Savinell RF, Wainright JS, Ray D (2009) 1H and 31 P NMR study of phosphoric acid-doped polybenzimidazole under controlled water activity. J Electrochem Soc 156(2):B283–B290

The openFuelCell project, http://openfuelcell.sourceforge.net/project

Wainright J, Wang J, Weng D, Savinell R, Litt M (1995) Acid-doped polybenzimidazoles: a new polymer electrolyte. J Electrochem Soc 142(7):L121–L123