Landes Bioscience www.landesbioscience.com

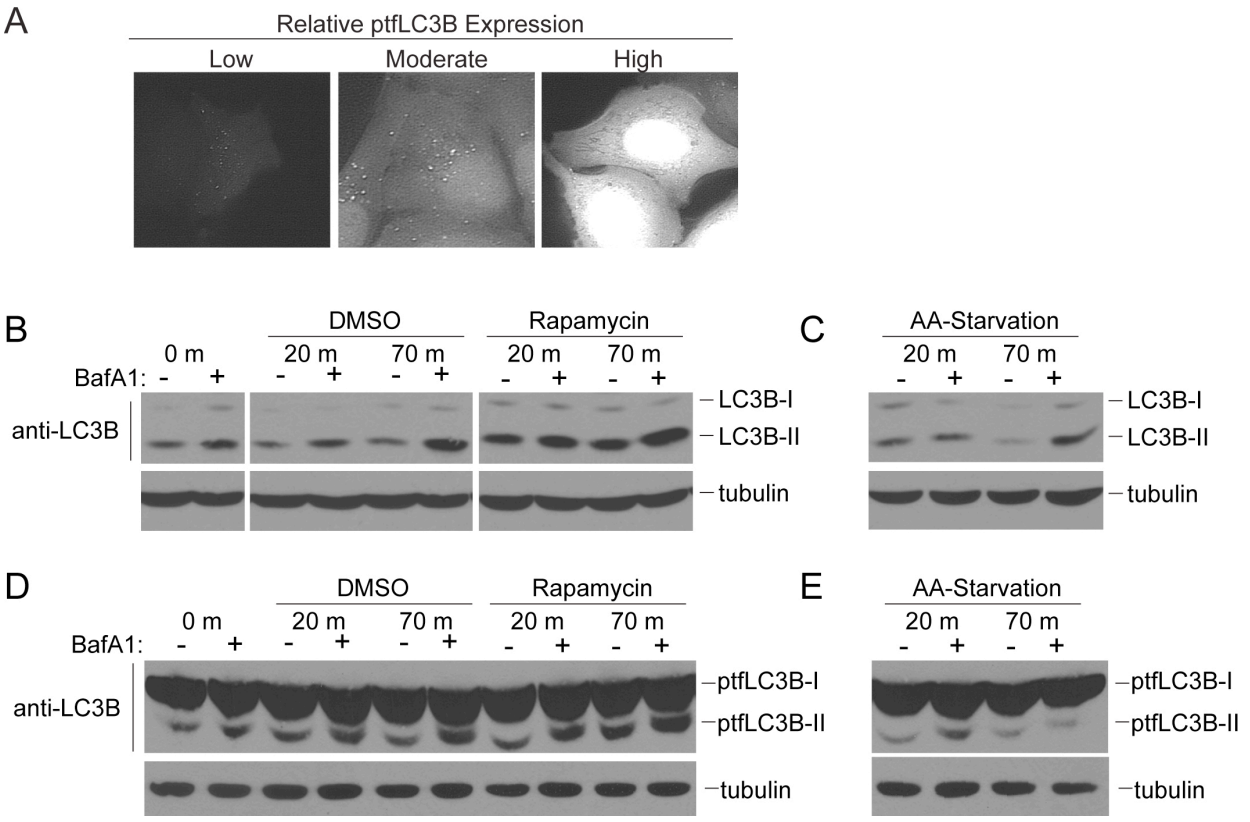
Supplemental Material to:

Katie R. Martin, Dipak Barua, Audra L. Kauffman, Laura M. Westrate, Richard G. Posner, William S. Hlavacek and Jeffrey P. MacKeigan

Computational model for autophagic vesicle dynamics in single cells

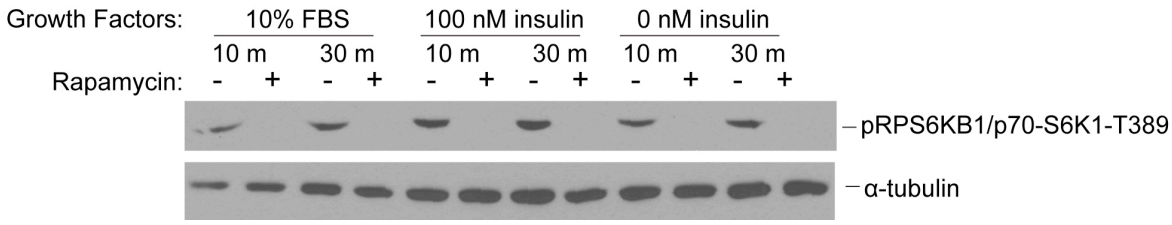
Autophagy 2012; 9(1)
http://dx.doi.org/10.4161/auto.22532

www.landesbioscience.com/journals/autophagy/article/22532

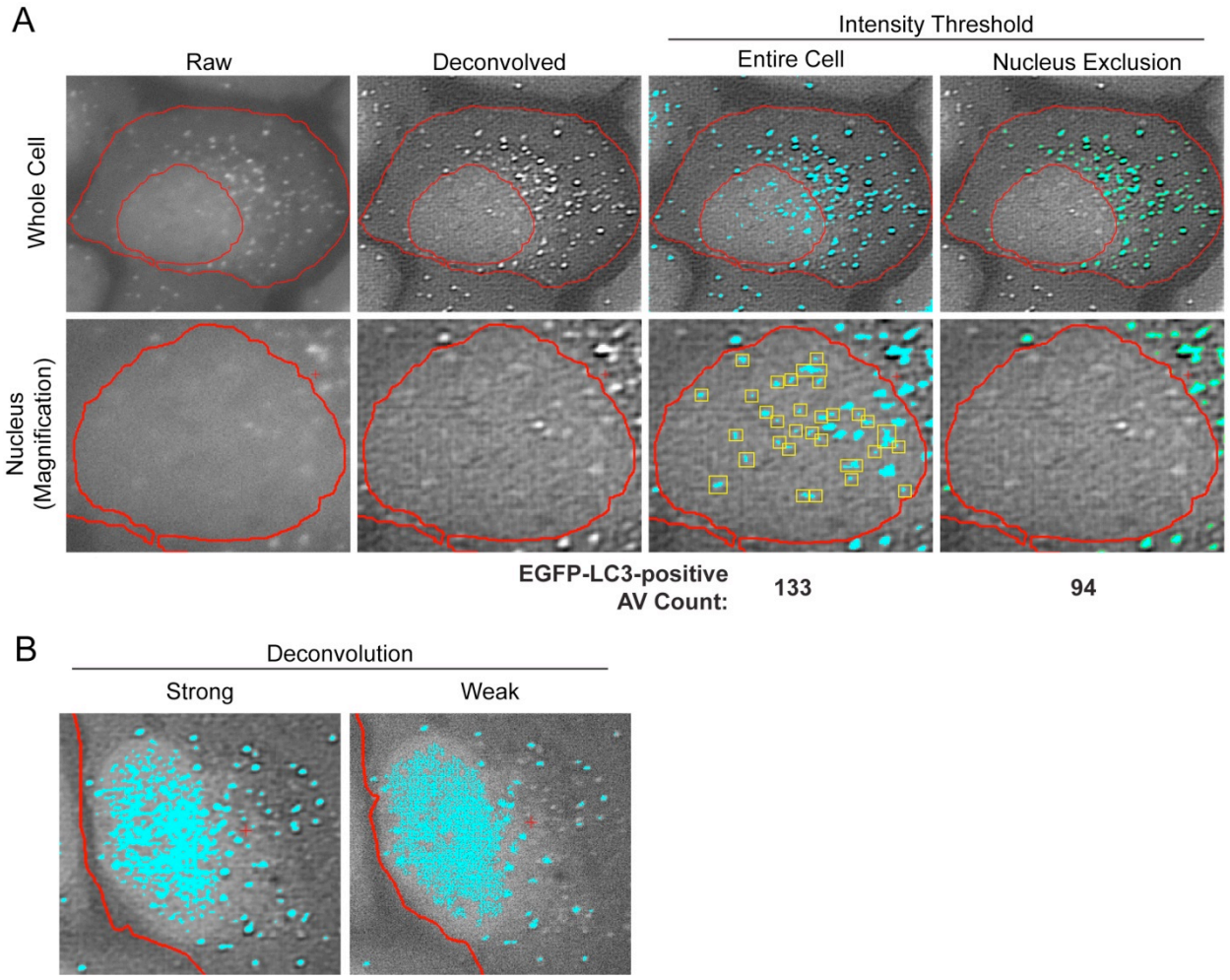


Supplemental Figure S1. Characterization of the monoclonal U2OS cell line stably expressing EGFP-LC3. **A**. EGFP-LC3 was imaged from stably-expressing cells (U2OS-ptfLC3B) demonstrating relatively low, moderate, or high expression. A monoclonal cell line displaying moderate expression was chosen and used for analysis within this study. **B,C.** Wild-type U2OS cells were treated with either rapamycin (100 nM) or DMSO control (B), or amino acid starvation media (C; 1 x DPBS with 10% FBS and 1g/L D-glucose) for 20 or 70 min., with (+) or without (-) 125 nM bafilomycin A_1 (BafA1). Each treatment included a 21 min pre-treatment with (+) or without (-) BafA1, as appropriate (pre-treatment only shown in left two wells of (B). Lysates were resolved by SDS-PAGE and immunoblotted with LC3B or alpha-tubulin antibodies, as indicated. **D,E.** The experiment was repeated using the monoclonal U2OS cell line stably expressing ptfLC3B plasmid. High molecular weight conjugates of RFP-GFP-LC3-I (~70 kDa) and RFP-GFP-LC3-II (~68 kDa) were detected using LC3 antibodies. Alpha-tubulin was detected as a loading control. Note the similar accumulation of LC3-II in response to BafA1 treatment or autophagy induction by rapamycin. Amino acid starvation induced

robust autophagic flux and turnover of LC3 (decreased LC3-II levels), an effect incompletely blocked by acute BafA1 treatment.

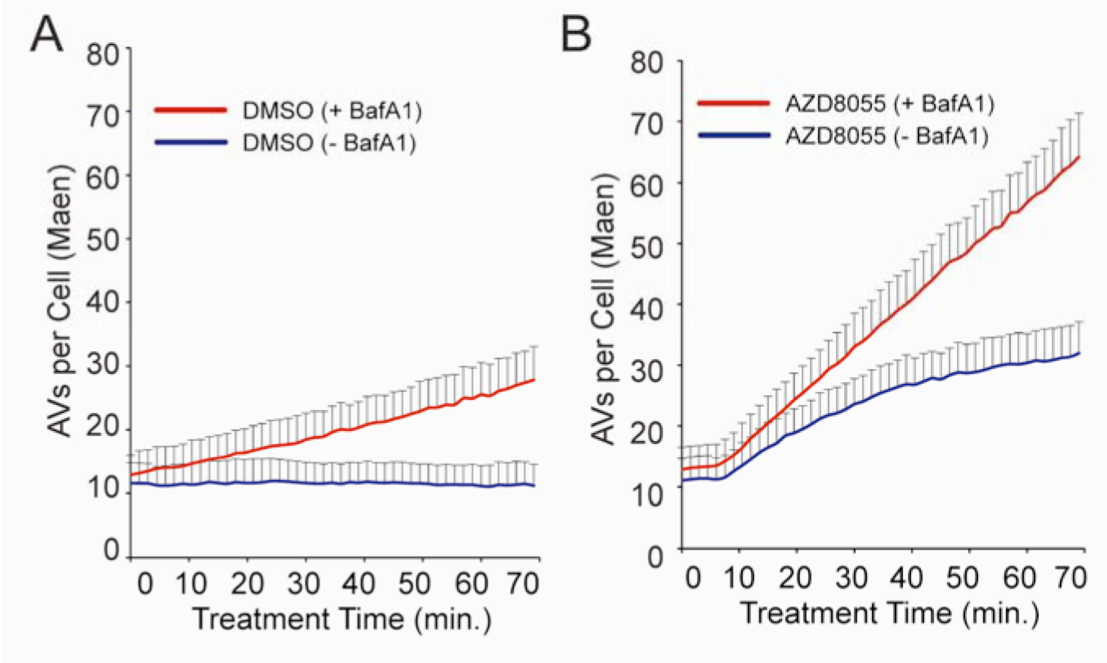


Supplemental Figure S2. The efficacy of rapamycin-mediated MTORC1 inhibition under various media conditions. 100 nM rapamycin was added for 10 or 30 min. to media containing 10% FBS (normal cell maintenance media), 100 nM insulin, or 0 nM insulin (insulin starvation). The treatment of insulin was identical to that used in quantitative autophagy experiments. Cell lysates were resolved by SDS-PAGE and immunoblotted using antibodies to detect phosphorylated RPS6KB1/p70-S6K1 at residue T389, an MTORC1-specific phosphorylation site. Alpha-tubulin was probed as a loading control. It can be noted that rapamycin was quite effective and ablated the phosphorylation of RPS6KB1 signal within 10 minutes, regardless of media composition. 100 nM insulin treatment modestly increased phopho-RPS6KB1 levels, an effect negated by insulin starvation.



Supplemental Figure S3. Image analysis procedure involved the exclusion of nuclei from adherent cell images. **A.** Images are of EGFP-LC3 puncta from a representative cell. From left-to-right: original (raw) image, the deconvolved image (using 2D blind deconvolution detailed in *Methods*), the cell with an intensity threshold set that includes the entire cell (detailed provided in *Methods*), the same intensity threshold set to exclude objects in the nucleus. Objects included in the thresholded region are in cyan and counted objects are outlined in green in the far right image. The region of interest used in quantification is drawn in red. The bottom panels are magnifications of the nuclear region. Note the inappropriate detection of false objects in the nucleus (yellow boxes) that did not represent true AVs as discriminated by manual inspection. A few objects may have represented real AVs (cyan objects within the nucleus, not boxed) but

these were far fewer in number than the inaccurately detected objects. The resulting EGFP-LC3 AV count from the cell with and without the nucleus included is shown. **B.** All combinations of deconvolution settings were attempted to reduce the fluorescence from the nucleus. The strongest deconvolution settings were chosen for use (left) and any combination, including settings with weaker stringency, did not alleviate the issue or improve the accuracy of puncta counts (right). Again, a portion of a representative cell is shown, the ROI border can be seen in red and the objected identified by the intensity threshold are shown in cyan.



Supplemental Figure S4. Stochastic simulation results, including standard deviations, for basal and induced autophagy. As in Figure 4, simulation results (blue curves) were generated for DMSO (A; basal autophagy) and AZD8055 (B; induced autophagy) treatments in the presence (+ BafA1; solid curves) and absence (- BafA1; broken curves) of BafA1. Curves represent averages of 100 independent stochastic simulation runs. Bars represent standard deviations. Each simulation run corresponds to the behavior predicted for a single cell.

Supplemental File 1. The source code for the mechanistic model of autophagy is provided in text format.

```
/* Model for autophagic vesicle dynamics in single cells */

/* Last updated on 05/10/2012 */


#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <time.h>


#define SimSize   50000


int Input=56;

int Vps34=100000;

int PI=1000000;

int WIPI=100000;

int Atg9=100000;

int species5=0;

int species6=0;

int species7=0;

int species8=0;

int species9=0;

int species10=0;

int species11=0;

int species12=0;

int species13=0;

int species14=0;

int species15=0;

int species16=0;

int species17=0;

int species18=0;

int species19=0;

int LC3=100000;


float kact=8.0e-8;

float kdea=0.1;

float kp1=1e-6;

float km1=0.1;

float kcat=1;

float pase=1;
```

```
float kcap=1e-6;

float krel=0.1;

float keng=1e-6;

float kdis=0.1;

float kfor=0.1;

float kpip2=1;

float kdpip2=1;

float nu=0.01;

float kflux=0.1;

float kfluxAtg4=0.1;

float kp=1e-6;

float km=0.1;

float kmat=1;

float k_ves=0.05;

float kagg=5e-4;

float ksplit=0.15;

float kdeg=4.5e-4;

float krela=1e-6;


/*--------Control parameters---------- */

int baf=1; /* binary parameter; 1= baf, no lysosomal degradation, 0=no baf, lysosomal degradation*/

float t_equil=10000;

float t_rapa=1260; /* time (sec) before azd addition, i.e., pretreatment time*/

float dose=2.619;  /* Input/azd dose; should be >=0; 0 represents the basal condition, i.e., not azd addition*/

int thold=400; /*Threshold z when IM converts into a phagophore/vesicle*/

/*-------------------------*/


int s[31][5]; /*Stoichiometric coefs; to be read from file core_ss.txt*/

int ss[9][4]; /*Stochiometric coefs for reaction in an IM; to be read from file ss_txt*/


int i,j,ii;

float n1, n2; /* Random numbers */

float a0=0;  /* sum of reaction propensities*/

float tau=0; /*Reaction time*/

float delt=0; /*Delay time for next reaction*/

float mu=0; /*Constant to determine what reaction will be next*/

float report_time=0;

int oldIM=0;

int newIM=0;

int vesicle=0;
```

```
struct species{
       int data;
       };

struct rxn{
       int flag;
       float ratek;
       float prop;
       int *coef;
       struct species*  link[4];
       struct rxn* next;
       struct rxn* prev;
        };

struct rxn* head;
struct rxn* ptr;
struct rxn* tptr;
struct rxn* temp;

FILE* output;
FILE* StoCof;
FILE* miniStoCof;

int SpcNum, RxnNum;
struct species* spc[1000];
struct species *y;
struct species *x;
struct species *xy;
struct species *z;
struct species *v;
struct species *Atg4;
struct species *blink;
struct species *marker;

void Core(), delete_vesicle(struct rxn* node), CoreRxnList(), VesicleRxnList(int IM), ReadRxnStoich();
int observables(float tstep);
double RAN();

main(){
     int sss;
     ReadRxnStoich();
```

```
    system("cp time.data temp0.data");
    for (sss=0;sss<1;sss++){
        tau=0;
        Core();
          system("awk '{print $3}' output.txt > temp1.data");
          system("paste temp0.data temp1.data > temp2.data");
          system("mv temp2.data temp0.data");
          printf("\n");
          }
          system("cp temp0.data sim_avg.data");
          /*system("bash gnuplot_combined_errorbar.sh");*/
   }
void Core(){
      head=malloc(sizeof(struct rxn));
      head->next=NULL;
      ptr=head;
      tptr=ptr;

      CoreRxnList(); /*Read core reaction list*/

      output=fopen("output.txt","w");

      y=malloc(sizeof(struct species));
      blink=malloc(sizeof(struct species));
      y->data=LC3;
      blink->data=0;

    /* Start Simulation run */



    report_time=0;
    while (tau<t_equil+t_rapa+4500) {
        if(tau < t_equil+t_rapa){
             spc[0]->data=Input;   // Basal condition;
             }
          else if (tau>=t_equil+t_rapa)
             {
             spc[0]->data=Input+dose*Input;   // Stimulation
             }
```

```
  n1=RAN();

n2=RAN();



/* Update vesicle list */

tptr=ptr;

  blink->data=0;



while(tptr->next){

       /*if (tptr->flag==1) {fprintf(output,"%d\n",tptr->link[0]->data);}*/

       if((tptr->flag==7)&&(tptr->prev->link[1]->data >=thold)){   // If contains >= theshold number of z molecules, membrane should have a positive rate to form closed vesicle.

           tptr->ratek=k_ves;

             }

       if((tptr->flag==7)&&(tptr->link[0]->data >=1)){  // If membrane is marked as 'matured' (indicated by 1), must be made a closed vesicle.

             //tptr->prev->flag=10; // To make Atg4-mediated LC3 dissoiation reaction active;

             tptr->prev->prev->flag=10;

             tptr->prev->prev->prev->flag=10;

             tptr->prev->prev->prev->prev->flag=10;

             tptr->prev->prev->prev->prev->prev->flag=10;

             tptr->prev->prev->prev->prev->prev->prev->flag=10;

             tptr->flag=10;

             tptr->prev->ratek=krela; // Make LC3 dissociation reaction active.

             tptr->prev->prev->ratek=0;

             tptr->prev->prev->prev->ratek=0;

             tptr->prev->prev->prev->prev->ratek=0;

             tptr->prev->prev->prev->prev->prev->ratek=0;

             tptr->prev->prev->prev->prev->prev->prev->ratek=0;

             tptr->ratek=0;

             tptr->next->flag=100;

             tptr->prev->ratek=krela; //Allow positive rate for Atg4-mediated z dissociation.

             if (tau<t_equil){

                 tptr->next->ratek=kdeg*(1+(0*(vesicle)/(10+vesicle)));

                 }

             else

                 {

                 tptr->next->ratek=kdeg*(1-baf)*(1+(0.0*(vesicle)/(10+vesicle)));

                 }

             }

       if((tptr->flag==100)&&(tptr->next->flag!=200)){

           tptr->next->flag=200;  //A closed vesicle (flag=100) must be subject to dimerization/splitting reaction (flag=200).

             tptr->next->link[0]->data=1;   // Newly formed closed vesicle must appear as a 'visible' vesicle (data=1).
```

```
                    }
               if((tptr->flag==200)&&(tptr->link[0]->data==1)){
                   blink->data=blink->data+1;  // Newly formed closed vescile adds one more 'visible' vesicle into the pool.
                    }
                    tptr=tptr->next;
                }



            tptr=ptr;
            while(tptr->next){
                 if((tptr->flag==200)&&(tptr->link[0]->data==1)){
                     tptr->ratek=kagg*(blink->data-1); // Dimerization rate is determined by the number number of 'visible' vesicles (a bimolecular
reaction).
                      }
                 else if ((tptr->flag==200)&&(tptr->link[0]->data==0)){
                     tptr->ratek=ksplit; //Splitting rate is determined by the number of 'invisible' vesicles (a unimolecular reaction).
                      }
                      tptr=tptr->next;
                   }

            /* Calculate reaction propensities*/
          a0=0;
            tptr=ptr;
            while(tptr->next){
                tptr->prop=tptr->ratek;
                ii=0;
                while(*(tptr->coef+ii)<0){
                       tptr->prop=tptr->prop*(tptr->link[ii]->data);
                       ii++;
                     }
                a0=a0+tptr->prop;
                tptr=tptr->next;
                }

         /*Calculate next reaction time*/
          delt=-(1/a0)*log(n1);
          tau=tau+delt;

          /*Select next reaction*/
            tptr=ptr;
          mu=0;
```

```
          mu=tptr->prop;

          while((mu < n2*a0)){

                 tptr=tptr->next;

                 mu=mu+tptr->prop;

                 }



          /* Update species*/

            temp=tptr;

             if (temp->flag==7){

                 temp->link[0]->data=1;  // Marked for vesicle closure/degradation.

                 temp=NULL;

                 free(temp);

                 }

             else if (temp->flag ==100){     // Degradation for corrresponding vesicle.

                 delete_vesicle(temp);

                 }

             else if(temp->flag==1){

                 temp->link[0]->data=temp->link[0]->data+1; // Constant flux of X into a vesicle.

                 temp=NULL;

                 free(temp);

                 }

             else if(temp->flag==5){

                 temp->link[0]->data=temp->link[0]->data+1;

                 temp=NULL;

                 free(temp);

                 }

             else if((temp->flag==200)&&(temp->link[0]->data==0)){

                 temp->link[0]->data=1;      // Selected reaction is splitting of a vesicle dimer; so mark the selected vesicle as 'visible'
(indicated by 1).

                 temp=NULL;

                 free(temp);

                 }

             else if((temp->flag==200)&&(temp->link[0]->data==1)){

                 temp->link[0]->data=0; // Selected reaction is dimerization of two vesicles; mark the selected vesicle as 'invisible' (indicated by
0).

                 temp=NULL;

                 free(temp);

                 }

             else{  //

                 ii=0;

               while(*(temp->coef+ii)){

                      temp->link[ii]->data=temp->link[ii]->data + *(temp->coef+ii);
```

```
                    ii++;
                    }
                temp=NULL;
                free(temp);
               }

          newIM=spc[5]->data;

          VesicleRxnList(newIM);
            if (newIM>0){
                spc[5]->data=spc[5]->data-1;
                newIM=spc[5]->data;
                }

            /* Count and report how many vescicles are there and their sizes */

            if ((tau >=t_equil+t_rapa*0)&&((tau-report_time) >=90)){
                report_time=tau;
                vesicle=observables(tau-(t_equil+t_rapa));
                }
      }
          fclose(output);

 }



int observables(float time){
     int vesicle;
     int IsoMem;
     tptr=ptr;
     vesicle=0;
     IsoMem=0;
     while(tptr->next){
            if (tptr->flag==200){
               vesicle++;
               }
            else if (tptr->flag==9){
               IsoMem++;
               }
               tptr=tptr->next;
```

```
        }

    fprintf(output,"%0.3f\t%d\t%d\t%d\t",time/60,vesicle,blink->data,IsoMem);
    printf("%0.3f\t%d\t%d\t%d\t\n",time/60,vesicle,blink->data,IsoMem);

    tptr=ptr;
    while(tptr->next){
       if (tptr->flag==100){
          fprintf(output,"%d   ",tptr->link[0]->data);
          }
          tptr=tptr->next;
        }
    fprintf(output,"\n");
    return (vesicle);
}


void VesicleRxnList(int IM){
      if (IM>0) {
           x=malloc(sizeof(struct species));
           xy=malloc(sizeof(struct species));
           z=malloc(sizeof(struct species));
           v=malloc(sizeof(struct species));
           Atg4=malloc(sizeof(struct species));
           marker=malloc(sizeof(struct species));

           x->data=0;
           xy->data=0;
           z->data=0;
           v->data=0;
           Atg4->data=0;
           marker->data=0;

           head->flag=1;
           head->ratek=kflux;
           head->prop=0;
           head->coef=ss[0];
           head->link[0]=x;
           head->next=malloc(sizeof(struct rxn));
           head->next->prev=head;
           head=head->next;
```

```
  head->flag=2;
  head->ratek=kp;
  head->prop=0;
  head->coef=ss[1];
  head->link[0]=x;
  head->link[1]=y;
  head->link[2]=xy;
  head->next=malloc(sizeof(struct rxn));
  head->next->prev=head;
  head=head->next;

head->flag=3;
  head->ratek=km;
  head->prop=0;
  head->coef=ss[2];
  head->link[0]=xy;
  head->link[1]=x;
  head->link[2]=y;
  head->next=malloc(sizeof(struct rxn));
  head->next->prev=head;
  head=head->next;

  head->flag=4;
  head->ratek=kmat;
  head->prop=0;
  head->coef=ss[3];
  head->link[0]=xy;
  head->link[1]=z;
  head->link[2]=x;
  head->next=malloc(sizeof(struct rxn));
  head->next->prev=head;
  head=head->next;

  head->flag=5;
  head->ratek=kfluxAtg4;
  head->prop=0;
  head->coef=ss[4];
  head->link[0]=Atg4;
  head->next=malloc(sizeof(struct rxn));
  head->next->prev=head;
```

```
  head=head->next;


  head->flag=6;
  head->ratek=0;
  head->prop=0;
  head->coef=ss[5];
  head->link[0]=head->prev->link[0];
  head->link[1]=head->prev->prev->link[1];
  head->link[2]=y;
  head->link[3]=head->prev->link[0];
  head->next=malloc(sizeof(struct rxn));
  head->next->prev=head;
  head=head->next;


head->flag=7;
  head->ratek=0.0;
  head->prop=0;
  head->coef=ss[6];
  head->link[0]=v;
  head->next=malloc(sizeof(struct rxn));
  head->next->prev=head;
  head=head->next;


  head->flag=8;
  head->ratek=0;
  head->prop=0;
  head->coef=ss[7];
  head->link[0]=head->prev->prev->prev->prev->link[1];
  head->next=malloc(sizeof(struct rxn));
  head->next->prev=head;
  head=head->next;


  head->flag=9;
  head->ratek=0;
  head->prop=0;
  head->coef=ss[8];
  head->link[0]=marker;
  head->next=malloc(sizeof(struct rxn));
  head->next->prev=head;
  head=head->next;
  head->next=NULL;
```

```
                        }
}

void CoreRxnList(){

              spc[0]=malloc(sizeof(struct species));
                spc[1]=malloc(sizeof(struct species));
              spc[2]=malloc(sizeof(struct species));
                spc[3]=malloc(sizeof(struct species));
              spc[4]=malloc(sizeof(struct species));
                spc[5]=malloc(sizeof(struct species));
              spc[6]=malloc(sizeof(struct species));
                spc[7]=malloc(sizeof(struct species));
              spc[8]=malloc(sizeof(struct species));  //PI3P
                spc[9]=malloc(sizeof(struct species));
              spc[10]=malloc(sizeof(struct species));
                spc[11]=malloc(sizeof(struct species));
              spc[12]=malloc(sizeof(struct species));
                spc[13]=malloc(sizeof(struct species));
                spc[14]=malloc(sizeof(struct species)); //PI3P2
                spc[15]=malloc(sizeof(struct species)); //(WIPI.PI3P2)
                spc[16]=malloc(sizeof(struct species)); //C0'
                spc[17]=malloc(sizeof(struct species)); //C1'
                spc[18]=malloc(sizeof(struct species)); //C2'
                spc[19]=malloc(sizeof(struct species)); //C3'


                spc[0]->data=Input;
                spc[1]->data=Vps34;
                spc[2]->data=PI;
              spc[3]->data=WIPI;
                spc[4]->data=Atg9;
                spc[5]->data=species5;
              spc[6]->data=species6;
                spc[7]->data=species7;
                spc[8]->data=species8;
              spc[9]->data=species9;
                spc[10]->data=species10;
                spc[11]->data=species11;
                spc[12]->data=species12;
                spc[13]->data=species13;
                spc[14]->data=species14;
```

```
  spc[15]->data=species15;
  spc[16]->data=species16;
  spc[17]->data=species17;
  spc[18]->data=species18;
  spc[19]->data=species19;



head->flag=0;
  head->ratek=kact;
  head->prop=0;
  head->coef=s[0];
  head->link[0]=spc[0];
  head->link[1]=spc[1];
  head->link[2]=spc[0];
  head->link[3]=spc[6];
  head->next=malloc(sizeof(struct rxn));
  head->next->prev=head;
  head=head->next;

  head->flag=0;
  head->ratek=kdea;
  head->prop=0;
  head->coef=s[1];
  head->link[0]=spc[6];
  head->link[1]=spc[1];
  head->next=malloc(sizeof(struct rxn));
  head->next->prev=head;
  head=head->next;

  head->flag=0;
  head->ratek=kp1;
  head->prop=0;
  head->coef=s[2];
  head->link[0]=spc[2];
  head->link[1]=spc[6];
  head->link[2]=spc[7];
  head->next=malloc(sizeof(struct rxn));
  head->next->prev=head;
  head=head->next;

  head->flag=0;
```

```
  head->ratek=km1;
  head->prop=0;
  head->coef=s[3];
  head->link[0]=spc[7];
  head->link[1]=spc[2];
  head->link[2]=spc[6];
  head->next=malloc(sizeof(struct rxn));
  head->next->prev=head;
  head=head->next;

head->flag=0;
  head->ratek=kcat;
  head->prop=0;
  head->coef=s[4];
  head->link[0]=spc[7];
  head->link[1]=spc[6];
  head->link[2]=spc[8];
  head->next=malloc(sizeof(struct rxn));
  head->next->prev=head;
  head=head->next;

  head->flag=0;
  head->ratek=pase;
  head->prop=0;
  head->coef=s[5];
  head->link[0]=spc[8];
  head->link[1]=spc[2];
  head->next=malloc(sizeof(struct rxn));
  head->next->prev=head;
  head=head->next;

  head->flag=0;
  head->ratek=kcap;
  head->prop=0;
  head->coef=s[6];
  head->link[0]=spc[3];
  head->link[1]=spc[8];
  head->link[2]=spc[9];
  head->next=malloc(sizeof(struct rxn));
  head=head->next;
```

```
head->flag=0;

head->ratek=krel;

head->prop=0;

head->coef=s[7];

head->link[0]=spc[9];

head->link[1]=spc[3];

head->link[2]=spc[8];

head->next=malloc(sizeof(struct rxn));

head=head->next;


head->flag=0;

head->ratek=keng;

head->prop=0;

head->coef=s[8];

head->link[0]=spc[4];

head->link[1]=spc[9];

head->link[2]=spc[10];

head->next=malloc(sizeof(struct rxn));

head=head->next;


head->flag=0;

head->ratek=kdis;

head->prop=0;

head->coef=s[9];

head->link[0]=spc[10];

head->link[1]=spc[4];

head->link[2]=spc[9];

head->next=malloc(sizeof(struct rxn));

head->next->prev=head;

head=head->next;


head->flag=0;

head->ratek=kfor;

head->prop=0;

head->coef=s[10];

head->link[0]=spc[10];

head->link[1]=spc[11];

head->next=malloc(sizeof(struct rxn));

head->next->prev=head;

head=head->next;
```

```
head->flag=0;

head->ratek=kdis;

head->prop=0;

head->coef=s[11];

head->link[0]=spc[11];

head->link[1]=spc[4];

head->link[2]=spc[9];

head->next=malloc(sizeof(struct rxn));

head->next->prev=head;

head=head->next;


head->flag=0;

head->ratek=kfor;

head->prop=0;

head->coef=s[12];

head->link[0]=spc[11];

head->link[1]=spc[12];

head->next=malloc(sizeof(struct rxn));

head->next->prev=head;

head=head->next;


head->flag=0;

head->ratek=kdis;

head->prop=0;

head->coef=s[13];

head->link[0]=spc[12];

head->link[1]=spc[4];

head->link[2]=spc[9];

head->next=malloc(sizeof(struct rxn));

head->next->prev=head;

head=head->next;


head->flag=0;

head->ratek=kfor;

head->prop=0;

head->coef=s[14];

head->link[0]=spc[12];

head->link[1]=spc[13];

head->next=malloc(sizeof(struct rxn));

head->next->prev=head;

head=head->next;
```

```
head->flag=0;

head->ratek=kdis;

head->prop=0;

head->coef=s[15];

head->link[0]=spc[13];

head->link[1]=spc[4];

head->link[2]=spc[9];

head->next=malloc(sizeof(struct rxn));

head->next->prev=head;

head=head->next;


head->flag=0;

head->ratek=nu;

head->prop=0;

head->coef=s[16];

head->link[0]=spc[13];

head->link[1]=spc[5];

head->link[2]=spc[10];

head->next=malloc(sizeof(struct rxn));

head->next->prev=head;

head=head->next;


head->flag=0;   //conversion of PI3P to PI3P2

head->ratek=kpip2;

head->prop=0;

head->coef=s[17];

head->link[0]=spc[8];

head->link[1]=spc[14];

head->next=malloc(sizeof(struct rxn));

head->next->prev=head;

head=head->next;


head->flag=0;   //conversion of PI3P2 to PI3P

head->ratek=kdpip2;

head->prop=0;

head->coef=s[18];

head->link[0]=spc[14];

head->link[1]=spc[8];

head->next=malloc(sizeof(struct rxn));

head->next->prev=head;
```

```
head=head->next;


head->flag=0;     // WIPI-PI3P2 binding

head->ratek=kcap;

head->prop=0;

head->coef=s[19];

head->link[0]=spc[3];

head->link[1]=spc[14];

head->link[2]=spc[15];

head->next=malloc(sizeof(struct rxn));

head->next->prev=head;

head=head->next;


head->flag=0;    //WIPI-PE3P2 dissociation

head->ratek=10*krel;

head->prop=0;

head->coef=s[20];

head->link[0]=spc[15];

head->link[1]=spc[3];

head->link[2]=spc[14];

head->next=malloc(sizeof(struct rxn));

head->next->prev=head;

head=head->next;


head->flag=0;   //Atg9-(WIPI.PI3P2) binding

head->ratek=keng;

head->prop=0;

head->coef=s[21];

head->link[0]=spc[4];

head->link[1]=spc[15];

head->link[2]=spc[16];

head->next=malloc(sizeof(struct rxn));

head->next->prev=head;

head=head->next;


head->flag=0;   //Atg9-(WIPI.PI3P2) dissociation

head->ratek=kdis;

head->prop=0;

head->coef=s[22];

head->link[0]=spc[16];

head->link[1]=spc[4];
```

```
head->link[2]=spc[15];

head->next=malloc(sizeof(struct rxn));

head->next->prev=head;

head=head->next;


head->flag=0;

head->ratek=kfor; //C0' formation

head->prop=0;

head->coef=s[23];

head->link[0]=spc[16];

head->link[1]=spc[17];

head->next=malloc(sizeof(struct rxn));

head->next->prev=head;

head=head->next;


head->flag=0;

head->ratek=kdis; //c0' disintegration

head->prop=0;

head->coef=s[24];

head->link[0]=spc[17];

head->link[1]=spc[4];

head->link[2]=spc[15];

head->next=malloc(sizeof(struct rxn));

head->next->prev=head;

head=head->next;


head->flag=0;     //c1' formation

head->ratek=kfor;

head->prop=0;

head->coef=s[25];

head->link[0]=spc[17];

head->link[1]=spc[18];

head->next=malloc(sizeof(struct rxn));

head->next->prev=head;

head=head->next;


head->flag=0;   //c1' disintegration

head->ratek=kdis;

head->prop=0;

head->coef=s[26];

head->link[0]=spc[18];
```

```
        head->link[1]=spc[4];
        head->link[2]=spc[15];
        head->next=malloc(sizeof(struct rxn));
        head->next->prev=head;
        head=head->next;

        head->flag=0;    //c2' formation
        head->ratek=kfor;
        head->prop=0;
        head->coef=s[27];
        head->link[0]=spc[18];
        head->link[1]=spc[19];
        head->next=malloc(sizeof(struct rxn));
        head->next->prev=head;
        head=head->next;

        head->flag=0;  //c2' disintegration
        head->ratek=kdis;
        head->prop=0;
        head->coef=s[28];
        head->link[0]=spc[19];
        head->link[1]=spc[4];
        head->link[2]=spc[15];
        head->next=malloc(sizeof(struct rxn));
        head->next->prev=head;
        head=head->next;

        head->flag=0;  //Nucleation of isolation membrane.
        head->ratek=nu;
        head->prop=0;
        head->coef=s[29];
        head->link[0]=spc[19];
        head->link[1]=spc[5];
        head->link[2]=spc[16];
        head->next=malloc(sizeof(struct rxn));
        head->next->prev=head;
        head=head->next;
        head->next=NULL;

}
```

```
void delete_vesicle(struct rxn* node){

     if (node->next){

     node->prev->prev->prev->prev->prev->prev->prev->prev->next=node->next->next;

     node->next->next->prev=node->prev->prev->prev->prev->prev->prev->prev->prev->prev;

     free(node);

     }

     }




void ReadRxnStoich(){

s[0][0]=-1;

s[0][1]=-1;

s[0][2]=1;

s[0][3]=1;


s[1][0]=-1;

s[1][1]=1;

s[1][2]=0;

s[1][3]=0;


s[2][0]=-1;

s[2][1]=-1;

s[2][2]=1;

s[2][3]=0;


s[3][0]=-1;

s[3][1]=1;

s[3][2]=1;

s[3][3]=0;


s[4][0]=-1;

s[4][1]=1;

s[4][2]=1;

s[4][3]=0;


s[5][0]=-1;

s[5][1]=1;

s[5][2]=0;

s[5][3]=0;


s[6][0]=-1;
```

```
s[6][1]=-1;

s[6][2]=1;

s[6][3]=0;


s[7][0]=-1;

s[7][1]=1;

s[7][2]=1;

s[7][3]=0;


s[8][0]=-1;

s[8][1]=-1;

s[8][2]=1;

s[8][3]=0;


s[9][0]=-1;

s[9][1]=1;

s[9][2]=1;

s[9][3]=0;


s[10][0]=-1;

s[10][1]=1;

s[10][2]=0;

s[10][3]=0;


s[11][0]=-1;

s[11][1]=1;

s[11][2]=1;

s[11][3]=0;


s[12][0]=-1;

s[12][1]=1;

s[12][2]=0;

s[12][3]=0;


s[13][0]=-1;

s[13][1]=1;

s[13][2]=1;

s[13][3]=0;


s[14][0]=-1;

s[14][1]=1;
```

```
s[14][2]=0;

s[14][3]=0;


s[15][0]=-1;

s[15][1]=1;

s[15][2]=1;

s[15][3]=0;


s[16][0]=-1;

s[16][1]=1;

s[16][2]=1;

s[16][3]=0;


s[17][0]=-1;

s[17][1]=1;

s[17][2]=0;

s[17][3]=0;


s[18][0]=-1;

s[18][1]=1;

s[18][2]=0;

s[18][3]=0;


s[19][0]=-1;

s[19][1]=-1;

s[19][2]=1;

s[19][3]=0;


s[20][0]=-1;

s[20][1]=1;

s[20][2]=1;

s[20][3]=0;


s[21][0]=-1;

s[21][1]=-1;

s[21][2]=1;

s[21][3]=0;


s[22][0]=-1;

s[22][1]=1;

s[22][2]=1;
```

```
s[22][3]=0;


s[23][0]=-1;

s[23][1]=1;

s[23][2]=0;

s[23][3]=0;


s[24][0]=-1;

s[24][1]=1;

s[24][2]=1;

s[24][3]=0;


s[25][0]=-1;

s[25][1]=1;

s[25][2]=0;

s[25][3]=0;


s[26][0]=-1;

s[26][1]=1;

s[26][2]=1;

s[26][3]=0;


s[27][0]=-1;

s[27][1]=1;

s[27][2]=0;

s[27][3]=0;


s[28][0]=-1;

s[28][1]=1;

s[28][2]=1;

s[28][3]=0;


s[29][0]=-1;

s[29][1]=1;

s[29][2]=1;

s[29][3]=0;


ss[0][0]=0;

ss[0][1]=0;

ss[0][2]=0;

ss[0][3]=0;
```

```
ss[1][0]=-1;

ss[1][1]=-1;

ss[1][2]=1;

ss[1][3]=0;


ss[2][0]=-1;

ss[2][1]=1;

ss[2][2]=1;

ss[2][3]=0;


ss[3][0]=-1;

ss[3][1]=1;

ss[3][2]=1;

ss[3][3]=0;


ss[4][0]=0;

ss[4][1]=0;

ss[4][2]=0;

ss[4][3]=0;


ss[5][0]=-1;

ss[5][1]=-1;

ss[5][2]=1;

ss[5][3]=1;


ss[6][0]=0;

ss[6][1]=0;

ss[6][2]=0;

ss[6][3]=0;


ss[7][0]=0;

ss[7][1]=0;

ss[7][2]=0;

ss[7][3]=0;


ss[8][0]=0;

ss[8][1]=0;

ss[8][2]=0;

ss[8][3]=0;

}
```

```
/*Long period (> 2 × 10^18) random number generator of L'Ecuyer with Bays-Durham shuffle
and added safeguards. Returns a uniform random deviate between 0.0 and 1.0 (exclusive of
the endpoint values). Call with idum a negative integer to initialize; thereafter, do not
alter idum between successive deviates in a sequence. RNMX should approximate the largest
floating value that is less than 1.*/
#define IM1 2147483563
#define IM2 2147483399
#define AM (1.0/IM1)
#define IMM1 (IM1-1)
#define IA1 40014
#define IA2 40692
#define IQ1 53668
#define IQ2 52774
#define IR1 12211
#define IR2 3791
#define NTAB 32
#define NDIV (1+IMM1/NTAB)
#define EPS 1.2e-7
#define RNMX (1.0-EPS)
long idum=-536445;
double RAN(){

    int j;
    long k;
    static long idum2=123456789;
    static long iy=0;
    static long iv[NTAB];
    float temp;
    if (idum <= 0){                          // Initialize.
        if (-(idum) < 1) idum=1;             // Be sure to prevent idum = 0.
        else idum = -(idum);
        idum2=(idum);
        for(j=NTAB+7;j>=0;j--){              // Load the shuffle table (after 8 warm-ups).
           k=(idum)/IQ1;
           idum=IA1*(idum-k*IQ1)-k*IR1;
           if (idum < 0) idum += IM1;
           if (j < NTAB) iv[j] = idum;
        }
        iy=iv[0];
```

```
    }
    k=(idum)/IQ1;                           // Start here when not initializing.
    idum=IA1*(idum-k*IQ1)-k*IR1;            // Compute idum=(IA1*idum) % IM1 without
    if (idum < 0) idum += IM1;              //     overflows by Schrage's method.
    k=idum2/IQ2;
    idum2=IA2*(idum2-k*IQ2)-k*IR2;          // Compute idum2=(IA2*idum) % IM2 likewise.
    if (idum2 < 0) idum2 += IM2;
    j=iy/NDIV;                              // Will be in the range 0..NTAB-1.
    iy=iv[j]-idum2;                         // Here idum is shuffled, idum and idum2 are
    iv[j] = idum;                           //     combined to generate output.
    if (iy < 1) iy += IMM1;
    if ((temp=AM*iy) > RNMX)
       return RNMX;                         // Because users don't expect endpoint values.
    else
       return temp;
}
```

Parameter	Sensitivity	Correlation
k_{flux}	0.956	+
LC3-I	0.629	-
k_{p}	0.593	-
k_{dis}	0.567	-
k_{deg}	0.498	-
k_{dpip2}	0.466	-
k_{mat}	0.331	-
k_{pip2}	0.263	-
PtdIns	0.231	+
k_{-1}	0.202	-
k_{rel1}	0.2	-
k_{rela}	0.192	-
I	0.19	+
k_{rel2}	0.17	-
k_{cat}	0.162	-
k_{nu}	0.159	+
k_{ves}	0.156	-
k_{+1}	0.148	+
k_{split}	0.139	-
ATG9	0.11	+
WIPI	0.1	+
k_{ase}	0.094	-
k_{cap}	0.093	+
PtdIns3KC3	0.089	-
k_{agg}	0.079	-
$k_{fluxATG4}$	0.061	-

k_m	0.042	+
k_{for}	0.028	-
k_{eng}	0.017	+
k_{dea}	0.0`3	-

Supplemental Table S1. Lag time sensitivity to model parameters. To assess lag time sensitivity to the model parameters, we systematically varied model parameter values and then evaluated corresponding effects on the lag time of the predicted AZD8055 (-BafA1) curve (Fig. 6). Lag time was defined as the time required to produce a certain number of vesicle counts after addition of AZD8055 in the model at time zero. After setting all parameters in the model to their nominal values (Table 1), we noticed a 17.5% increase in average vesicle count after 10 minutes of AZD8055 addition. We took this 17.5% increase as a standard, and evaluated how long it took to reach this level of increase when a particular parameter in the model was changed by a small amount. We systematically reduced each parameter value by 10%, while keeping the others fixed at nominal values, and noted the average time needed to reach the 17.5% increase in vesicle count. By applying the following equation, we then quantified the sensitivity of lag time to each parameter:

$$\text{Sensitivity} = \frac{\bar{P}_i}{\bar{\tau}} \frac{\Delta\tau_i}{\Delta P_i}$$

Here, $\bar{P}_i$ represents the nominal value for parameter i, $\bar{\tau}$ represents the lag time when all parameters are set to the nominal values ($\bar{\tau}$ ~ 10 min), ΔP_i represents the change in value of parameter i ($\Delta P_i = 0.1 P_i$), and $\Delta\tau_i$ represents the change in lag time due to the change in parameter i. The table shows the model parameters and corresponding lag time sensitivity (parameters responsible for higher sensitivity are shown in the top rows).

	Number of adjustable parameter (*k*)		$AIC = x^2 + 2k$	
	Deterministic	Stochastic	Deterministic	Stochastic
DMSO (-BafA1)	3	13	9.0 = 3+6	28.1=2.1+26
DMSO (+BafA1)	2	13	4.6 = 0.6+4	26.4=0.4+26
AZD8055 (-BafA1)	3	13	6.8=0.8+6	26.1=0.1+26
AZD8055 (+BafA1)	2	13	4.3=0.3+4	26.3=0.3+26

Supplemental Table S2. Goodness of fit for the deterministic and stochastic model. We applied the Akaike Information Criterion (AIC) to evaluate the relative goodness of fit for the deterministic model and the stochastic model. The two models were evaluated against the experimental data for the four treatment conditions: DMSO (-BafA1), DMSO (+BafA1), AZD8055 (-BafA1) and AZD8055 (-BafA1). Calculations were based on the fitting of the two models to the four datasets (Fig. 3 and Fig. 6). The following equation for AIC was applied:

$$AIC = \sum_{i=1}^{n} \frac{(y_i - f(x_i))^2}{\sigma_i^2} + 2k,$$

where, y_i is the mean experimental data (mean from n independent measurements) at a given time point x_i, and $f(x_i)$ is the corresponding model prediction. σ_i is the standard deviation of the data (at time x_i), and k is total number of adjustable parameters in the model. The deterministic model represents a better quality fit based on its smaller AIC values in all four conditions. The difference in the AIC values between the two models are primarily from their difference in number of adjustable parameters: whereas fitting of the deterministic model involved only 2 or 3 variable parameters (depending on the dataset), fitting of the stochastic model involved a total of 13 variable parameters. Therefore, the stochastic model acquires a large penalty in its AIC calculation from the greater number variable parameters.

Supplementary Movie 1. U2OS-EGFP-LC3 dynamics. A U2OS-EGFP-LC3 cell was imaged for 12 minutes at 10-second intervals. The green channel is displayed in black and white for easier visualization of EGFP-LC3 puncta. Note that individual puncta dipped slightly in and out of the focal plane and that distinct puncta appeared to merge and then break apart when in close proximity.