```
function [out_struct,busy_periods] = ggisr_retry_stats_hack(calls, paths, ss, si, tran
ge)
% function [out_struct,busy_periods] = ggisr_retry_stats_hack(calls, paths, ss, si, tr
ange)
%
% analyze the sample paths given by ggisr_retry_sim
% "ss" is not used at all.
% Fields: tries, wq, wo, w, visits,
% pr_loss, pr_balk, pr_delay,
% Lo, Lsq, Lq
% for example, out_struct.Lsq(1) = E[# in service+queue],
% out_struct.Lsq(2) = StdDev(# in service+queue)
% which is NOT the same as StdErr !
% trange is optional; it may be a single value, in which case it is tmin,
% or it may be a vector [tmin tmax]
% ASSUMES NO REVISITS!
% ss is currently not used.
% si is only barely used.

if( nargin <= 4 )
        trange(1) = min(min(paths.epochs),min(calls.arriv));
        trange(2) = max(max(paths.epochs),max(calls.arriv));
else
        if(length(trange) == 1)
                trange(2) = max(max(paths.epochs),max(calls.arriv));
        end
end

os.trange = trange;

mine=         calls.arriv >= trange(1) ;
mine = mine & calls.arriv <= trange(2) ;
% The following two should be equivalent
%mine_retried = mine & calls.n_tries > 1;
mine_retried = mine(:) & calls.wo(:) > 0;

ncalls = sum(mine);
os.ncalls = ncalls;

% record various input parameters,
% but don't choke if they don't exist.
fieldnames = {'lambda','mu','arrrate','svcrate','retrate',...
        'arrdist','svcdist','retdist',...
        'servers','nservers','room','buf','nbuf'};
for nf=1:length(fieldnames)
        if( isfield(si,fieldnames{nf}) )
                os.(fieldnames{nf}) = si.(fieldnames{nf});
        end
end
% Record the observed values of the usual input parameters:
% arrival rate, service durations, retrial durations
iats = diff(sort(calls.arriv(mine)));
os.iat(1) = nanmean(iats);
os.iat(2) = nanstd(iats);
os.svc(1) = nanmean(calls.svc_dur(mine));
os.svc(2) =  nanstd(calls.svc_dur(mine));

% Since we're only trying to estimate the input parameters,
% we can look at the whole list of potential retrials, rather than
% just the ones that actually happened, which is more complicated.
% We do want to throw out any negative values, though.
tmp = reshape(calls.ret_durs(mine,:),1,[]);
tmp = tmp(tmp>0);
os.ret(1) = nanmean(tmp);
os.ret(2) =  nanstd(tmp);
```

```
% The old way, using the calls.woa{} structure, we only recorded
% the retrials that actually happened:
%os.ret(1) = nanmean([calls.woa{mine}]);
%os.ret(2) =  nanstd([calls.woa{mine}]);
% But now, using calls.ret_durs(,) it records all the potential retrials
% and we want to filter out the ones that didn't happen
% First, let's only look at the calls
%tmp_ret = calls.ret_durs(mine_retried,:);
%for nc=1:size(tmp_ret)
%       first_invalid = calls.n_tries(nc);
%       tmp_ret(nc, (first_invalid+1):end) = NaN;
%end

os.ret_scv = (os.ret(2) / os.ret(1))^2; % usually 0, 1, 1/2, etc.

% This is how we would do it if data for one chain of calls
% wasn't distributed across the various call records.
% Since this is the "hack" file, this is how we'll do it.
% We don't care about n_visits, since we will assume no revisits.

tmp = calls.n_tries(mine);
os.tries(1)  = nanmean(tmp);
os.tries(2)  =  nanstd(tmp);
% Here, the 5 means 5-value summary:
% [min, quartile, median, quartile, max]
os.tries5(1) =  nanmin(tmp);
os.tries5(2:4) = prctile(tmp,[ 25 50 75]); % quartiles and median
os.tries5(5) =  nanmax(tmp);
% # of tries, conditional on at least one retrial
if( sum(mine_retried)>2 )
        tmp = calls.n_tries(mine_retried);
        os.tries_cond(1)  = nanmean(tmp);
        os.tries_cond(2)  = nanstd(tmp);
        os.tries_cond5(1) =  nanmin(tmp);
        os.tries_cond5(2:4) = prctile(tmp,[ 25 50 75]); % quartiles and median
        os.tries_cond5(5) =  nanmax(tmp);
else
        os.tries_cond5 = [0,0,0,0,0];
end
os.visits(1) = nanmean(calls.n_visits(mine));
os.visits(2) =  nanstd(calls.n_visits(mine));
os.wq(1) = nanmean(calls.wq(mine));
os.wq(2) =  nanstd(calls.wq(mine));
os.wo(1) = nanmean(calls.wo(mine));
os.wo(2) =  nanstd(calls.wo(mine));
% Time in orbit, conditional on at least one retrial
os.wo_cond(1) = nanmean(calls.wo(mine_retried));
os.wo_cond(2) = nanstd(calls.wo(mine_retried));
os.w(1)  = nanmean(calls.wo(mine) + calls.wq(mine) + calls.svc_dur(mine));
os.w(2)  =  nanstd(calls.wo(mine) + calls.wq(mine) + calls.svc_dur(mine));
% here we are counting revisits just like any other calls:
% Loss: call never gets served
os.pr_loss = sum(calls.n_visits(mine)==0) / ncalls;
% Balk: call entered orbit at least once
os.pr_balk = sum(calls.n_tries(mine)>1) / ncalls;
% Delay: call didn't enter service immediately.
os.pr_delay = sum((calls.wo(mine)+calls.wq(mine))>0) / ncalls;
% First success: got into service (or queue) on first try,
% which PASTA should apply to if arrivals are Poisson.
os.pr_first_success = sum(calls.n_tries(mine)==1) / ncalls;
os.pr_first_fail = 1-os.pr_first_success;
% Pct of retries that succeed/fail
tot_retries = sum(calls.n_tries(mine_retried) - 1);
tot_failed = sum(calls.n_tries(mine_retried) - 2);
os.pr_retry_fail = tot_failed / tot_retries;
```

```
% and now for the path-based stuff
mine=         paths.epochs >= trange(1) ;
mine = mine & paths.epochs <= trange(2) ;

durs = diff(paths.epochs(mine));
tspan = trange(2) - trange(1);


tmp = paths.n_orbit(mine);
tmp = tmp(1:end-1);
os.Lo(1) = sum(durs.*tmp    )/tspan;
tmp2     = sum(durs.*tmp.^2 )/tspan;%2nd moment
os.Lo(2) = sqrt(tmp2 - os.Lo(1)^2); % sqrt(variance)

tmp = paths.n_sq(mine);
tmp = tmp(1:end-1);
os.Lsq(1) = sum(durs.*tmp    )/tspan;
tmp2      = sum(durs.*tmp.^2 )/tspan;%2nd moment
os.Lsq(2) = sqrt(tmp2 - os.Lsq(1)^2); % sqrt(variance)

tmp = paths.n_sq(mine)+paths.n_orbit(mine);
tmp = tmp(1:end-1);
os.L(1) = sum(durs.*tmp    )/tspan;
tmp2    = sum(durs.*tmp.^2 )/tspan;%2nd moment
os.L(2) = sqrt(tmp2 - os.L(1)^2); % sqrt(variance)

tmp = max(0,paths.n_sq(mine)-si.nservers);
tmp = tmp(1:end-1);
os.Lq(1) = sum(durs.*tmp    )/tspan;
tmp2     = sum(durs.*tmp.^2 )/tspan;%2nd moment
os.Lq(2) = sqrt(tmp2 - os.Lq(1)^2); % sqrt(variance)

% Information on various arrival streams
% Probably don't need "nan"mean, but doesn't hurt to have it.
tmp = diff(paths.epochs(mine(:)' & paths.etypes=='a'));
os.streama(1) = nanmean(tmp);
os.streama(2) = nanstd(tmp);
os.streama_acf = autocorr(tmp)';
if( any(paths.etypes == 'r') )
tmp = diff(paths.epochs(mine(:)' & paths.etypes=='r'));
os.streamr(1) = nanmean(tmp);
os.streamr(2) = nanstd(tmp);
os.streamr_acf = autocorr(tmp)';
tmp = diff(paths.epochs(mine(:)' & (paths.etypes=='a' | paths.etypes=='r')));
os.streamb(1) = nanmean(tmp);
os.streamb(2) = nanstd(tmp);
os.streamb_acf = autocorr(tmp)';
end % if( any(paths.etypes == 'r') )
tmp = diff(paths.epochs(mine(:)' & paths.etypes=='s'));
os.streams(1) = nanmean(tmp);
os.streams(2) = nanstd(tmp);
os.streams_acf = autocorr(tmp)';

% And look at the marginal distributions at arrival times
% If we just use the system state at the arrival epoch, the new arrival
% is included, which we don't want.  Instead, we should look at the
% system state just before the arrival epoch, which is the same as the
% system state just after the previous event.
% So instead of using arr_mine, we should use arr_mine(2:end)
arr_mine = mine(:)' & paths.etypes=='a';
% accumarray is the nice way to do a discrete histogram, replacing "the sparse trick"
% but it must take strictly positive indices (zeros aren't allowed),
% so we add 1 to the # in service or orbit
os.arrp.p_nsq = accumarray(1+paths.n_sq(arr_mine(2:end)), 1);
```

```
os.arrp.p_no  = accumarray(1+paths.n_orbit(arr_mine(2:end)), 1);
os.arrp.corr = corr(paths.n_sq(arr_mine(2:end)),paths.n_orbit(arr_mine(2:end)));

ret_mine = mine(:)' & paths.etypes=='r';
if( sum(ret_mine) > 0 )
        os.retp.p_nsq = accumarray(1+paths.n_sq(ret_mine(2:end)), 1);
        os.retp.p_no  = accumarray(1+paths.n_orbit(ret_mine(2:end)), 1);
        os.retp.corr = corr(paths.n_sq(ret_mine(2:end)),paths.n_orbit(ret_mine(2:end))
);
end

% Analyze busy and idle periods
% busy=all servers busy, idle=at least one server idle
% If service is Exponential, and there's no organized queue,
% busy periods should be ~Exp(#servers*svcrate)
% Also, we think that long busy periods allow calls to accumulate in orbit,
% thus causing shorter subsequent idle periods.
% I could try to do this by vectorizing, but a for-loop seems easier.
nrows = 0;
busy_periods = NaN*[1,1;1,1];
for ni=min(find(mine)):max(find(mine))
        if( paths.n_sq(ni) >= si.nservers & paths.n_sq(ni-1) < si.nservers )
                % here's the start of a busy period
                nrows = nrows+1;
                busy_periods(nrows,1) = paths.epochs(ni);
                busy_periods(nrows,2) = NaN; % in case this busy period gets cut off
        elseif( nrows>0 & paths.n_sq(ni) < si.nservers & paths.n_sq(ni-1) >= si.nserve
rs )
                % here's the end of a busy period
                busy_periods(nrows,2) = paths.epochs(ni);
        end
end
busy_durs = busy_periods(:,2) - busy_periods(:,1);
os.busy(1) = nanmean(busy_durs);
os.busy(2) = nanstd(busy_durs);
idle_durs = busy_periods(2:end,1) - busy_periods(1:(end-1),2);
os.idle(1) = nanmean(idle_durs)
os.idle(2) = nanstd(idle_durs)
% for the correlation, we trim the last element because it likely has a NaN
os.busy_idle_corr = corr(busy_durs(1:(end-2)),idle_durs(1:(end-1)));
% Now for a non-intuitive estimator of blocking probability:
% use Little's Law L=lambda_a * W
% lambda_a = lambda *(1- Pr{block})
% using the _known lambda_ from the simulation parameters, not estimated
% from data.
% so Pr{block} = 1- L/(W*lambda)
% See "Simulation Run Lengths to Estimate Blocking Probabilities",
% Srikant and Whitt, 1996, ACM Transactions on Modeling and Computer Simulation
% Tends to be negatively correlated with (# blocked calls)/(# total calls)
%os.lambda = 1/si.a_dist;
% Needs more thought.
%os.pr_loss_2 = 1-os.Lsq(1) / ( os.w(1) * os.lambda );

out_struct = os;

return;

% if you want to plot a real (squared-off) sample path,
% let
 mine = 3:19;
% or
mine = 1:length(paths.epochs); % to get the full sample path
len = length(mine);
tmpe = paths.epochs(mine);
tmpn = paths.n_sq(mine);
```

```
% or
tmpn = paths.n_orbit(mine);
plot([reshape([tmpe, tmpe]',2*len,1);NaN],[NaN;reshape([tmpn,tmpn]',2*len,1)],'-+')
```