Paternoster policing and scheduling

A simple real-time packet bandwidth reservation, policing, queuing, and transmission scheduling algorithm that provides bounded network delays without requiring clock synchronization between adjacent network nodes.

.../docs2017/cr-seaman-paternoster-policing-scheduling-0317-v03.pdf

This presentation is intended to provide a brief overview. See the link above for detail.

Mick Seaman

3/14/2017

Objectives

- No time sync requirement node to node within the network
- Bounded delays/lossless for conformant flows

  (in presence of non-conformant flows)
- Easy to implement/understand/calculate

Basics

- All nodes use same epoch duration τ for given class of service
- End station/port can send a reservations worth of traffic (anywhere) in each τ period (for that station/port)
- Bridge port (for transmission)
  - Four queues per class of service per port
  - Each associated with an epoch (on a cyclic basis), identified (at a given time) as:

    prior, *current*, *next*, *last*

    (at next tick will become *last, prior, current, next*)
- On reception, identify reservation for frame, and add it to:

  current, but if doesn't fit in remaining reservation, add to

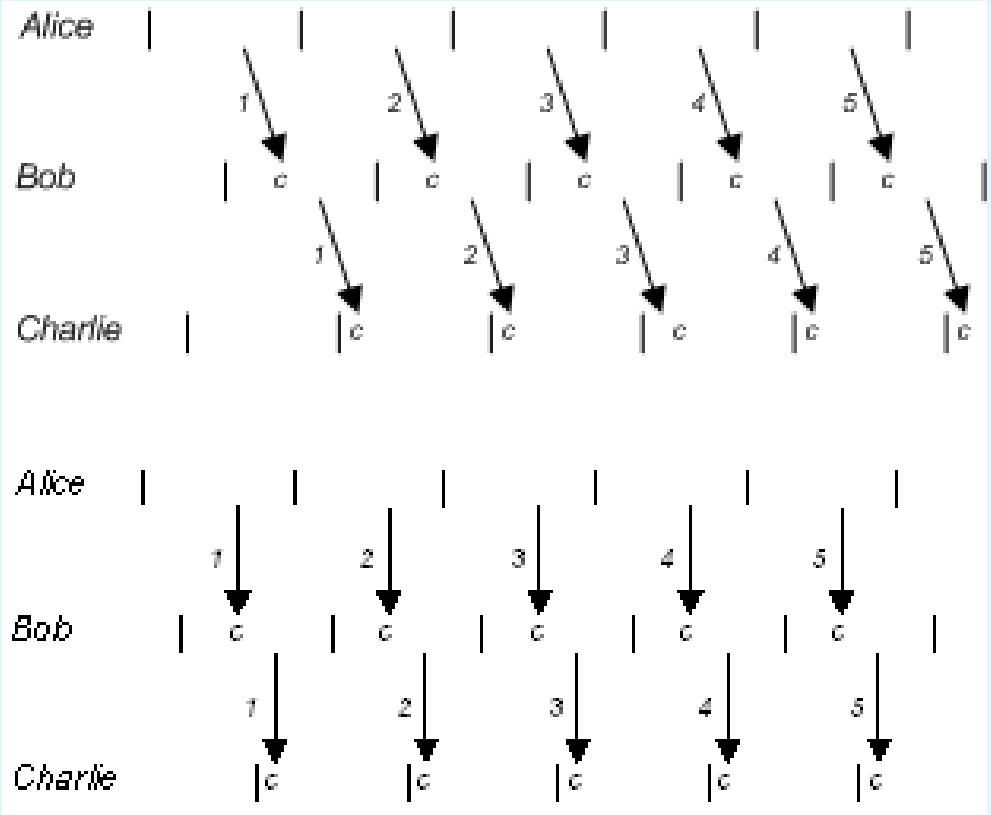
  next, but if doesn't fit in remaining reservation, add to

  last, but if it doesn't fit in remaining reservation, discard
- On transmission opportunity, transmit from:

  prior, if there is anything left on that queue, then

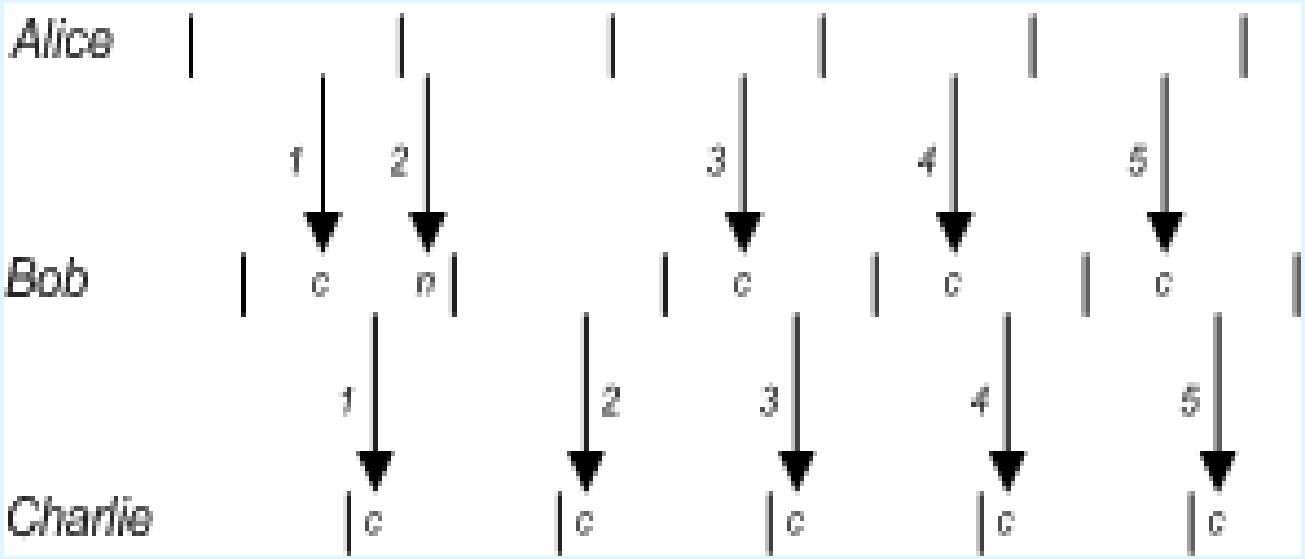
  current, if there is anything, on that queue

Some timing diagrams



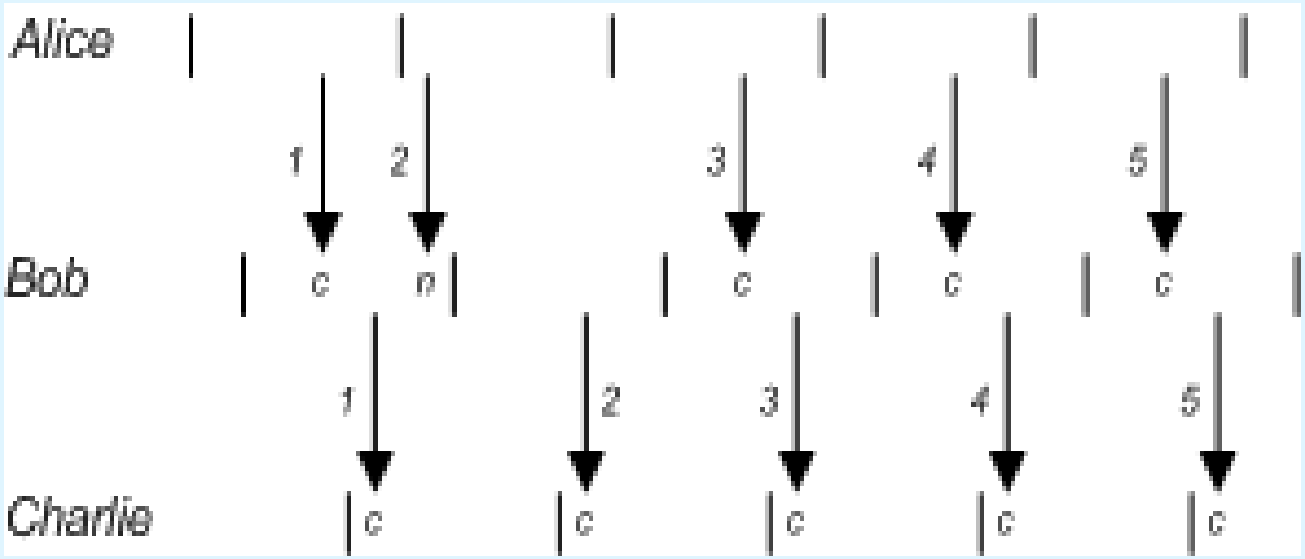
Time shift by minimum delay

Some timing diagrams



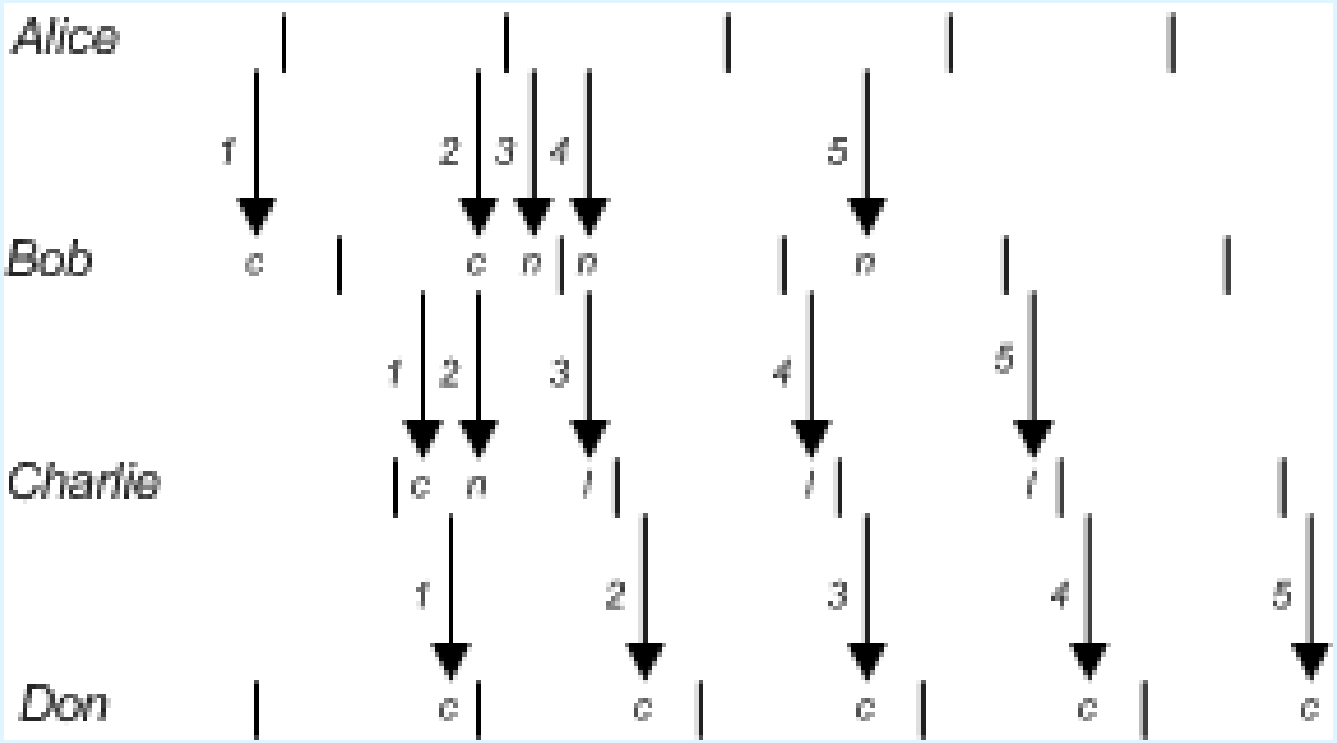
Can receive two epochs worth of a reservation in one epoch

Some timing diagrams



Can receive two epochs worth of a reservation in one epoch

Some timing diagrams



Can receive three epochs worth of a reservation in one epoch, but no more , provided transit delay variation + time to transmit prior < τ

Algorithm attributes

- Max node delay is 3 τ
- Supports preemption
- Supports layered service classes with shorter high priority epochs
- Supports long/high delay w/o changing local epoch timing by using more queues
- Independent epoch for transmission ports

  No hard sync requirement across the bridge

Pseudo-code (data types/structures)

```
#define Epochs 4 // epochs and transmit queues for each port and class
#define Reservations // number (arbitrary) of reservations per port and class

typedef Int Epoch; // {Zero, One, Two,. Queues-1} repeating
typedef Int Allocation;

typedef struct
{
Epoch prior;
Epoch current;
Epoch last;
Epoch tx;
} Port_class_epoch;

typedef struct
{
Epoch queue_for;
Allocation remaining;
Allocation permitted;
} Reservation;

Epoch following[Epochs];
Port_class_epoch epoch[Ports][Classes];
Queue queue[Ports][Classes][Epochs];
Reservation reservations[Ports][Classes][Reservations];
```

Pseudo-code (queuing relayed packet)

```
Boolean relay(port_no, class, reservation, packet, packet_allocation)
Port_no port_no;
Class class;
Reservation *reservation;
Packet packet;
Allocation packet_allocation;
{
Allocation remainder;
for(;;)
{
remainder = reservation->remaining - packet_allocation;
if ((remainder >= 0)
{
enqueue_packet(port_no, class, reservation->queue_for);
}
if ((remainder > 0) ||
(reservation->queue_for == epoch[port_no][class].last))
{
reservation->remaining = remainder; return (remainder >= 0);
}
reservation->queue_for = following[queue_for];
reservation->remaining = permitted;
if (remainder == 0)
{
return (remainder >= 0);
} } }
```

Pseudo-code (transmit selection)

```
Packet tx_select(port_no, class)
Port_no port_no;
Class class;
{
Packet packet;
for(;;)
{
packet = dequeue(port_no, class, epoch[port_no][class].tx);
if ((packet != Ptr_to_null) ||
(epoch[port_no][class].tx == epoch[port_no][class].current))
{
return(packet);
}
epoch[port_no][class].tx = epoch[port][class].current;
} } }
```

Pseudo-code (epoch updating)

```
epoch_tick(port_no, class)
Port_no port_no;
Class class;
{
Epoch temp = epoch[port_no][class].prior;
purge_queue(port_no, class, epoch[port_no][class].prior);
epoch[port_no][class].prior = following([port_no][class].prior);
epoch[port_no][class].current = following([port_no][class].current);
epoch[port_no][class].last = following([port_no][class].last);
for(i = 0; i < Reservations; i++)
{
if (reservations[port_no][class][i].queue_for !=
epoch[port_no][class].current)
{
reservations[port_no][class][i].queue_for =
epoch[port_no][class].current;
reservations[port_no][class][i].remaining =
reservations[port_no][class][i].permitted;
} } }
```