Computer Engineering 4DK4
Lab 2 Guidelines and Suggestions

Here are some suggestions and guidelines for Lab 2. Feel free to contact me or the TAs if you have any questions about anything. T. Todd.

1 Experiments

Part 3: Modify the simulation to keep track of the number of packets over a simulation run whose delay exceeds 40 msec, i.e., increment a counter for each packet whose delay exceeded 40 msec. Then output the fraction of those packets of the total when the simulation finishes.

Do simulation runs and manually search for a value of λ so that this fraction is 2%. You do not need to write code that automatically searches for this value of λ.

Part 4: Use the supplied code as a template and create three links and buffers, e.g., define them in `main.h` and declare and initialize them in `main.c`. The most obvious approach is to duplicate the functions

`schedule_packet_arrival_event`,
`packet_arrival_event`,
`schedule_end_packet_transmission_event`
and `end_packet_transmission_event` for the new events. For example,
`schedule_packet_arrival_to_S1_event`,
`packet_arrival_to_S1_event`,
`schedule_packet_arrival_to_S2_event`,
`packet_arrival_to_S2_event`, `schedule_packet_arrival_to_S3_event`,
`packet_arrival_to_S3_event`, etc.

Then edit the code to achieve the correct switch-to-switch routing, e.g., an end packet transmission event on S1 will place the transmitted packet in the buffer or link at S2 or S3.

If you are more adventurous, you could do everything using a single set of packet arrival and end packet transmission functions. You just need to identify which switch the packet is currently at.

In main.c You will need to initialize three packet arrival events, one each for S1, S2 and S3.

Add a member to the packet struct (in main.h) that records what switch the packet originally arrived to. When the packet is finished transmitting on S2 or S3, this is used to record the delay for those packets.

Part 5: Create a new arrival event for voice, `voice_packet_arrival_event`, i.e., copy `packet_arrival.h` and `packet_arrival.c` to `voice_packet_arrival.h` and `voice_packet_arrival.c`. Make the minor changes needed, e.g., voice packet arrivals have deterministic inter-arrival times, etc. You need to schedule the first voice packet arrival event in `main.c` (as is done for the data packet arrivals).

In `main.h` you should add a struct member to `Packet` so that you can mark it as VOICE or DATA. You need to add variables to `Simulation_Run_data` to collect separate voice packet counts and delay, etc. In `end_packet_transmission_on_link`, collect the required statistics.

Part 6: Use the separate voice/packet arrival events from Part 5. Create a separate voice buffer in `main.h`, e.g., `Fifoqueue_Ptr voice_buffer;` in `Simulation_Run_Data`. Initialize it in `main.c`. When a voice or data packet arrival occurs and the link is busy, place the packet in the appropriate data or voice buffer.

When there is an end of packet transmission event, it means that the link has become free. Always check the voice packet buffer for packets to start transmission first, followed by the data packet buffer. When a packet is found, start its transmission as usual.