Critique 1

COS Event Service described in the paper is actually a centralized broker system. Multiple consumers and producers connect to the event channel. Consumer constantly poll msgs while producers periodically push msgs to the event channel. One of the key point in this design is that the producers and consumers may never know each other. So the event channel can also be called broker, which acts as a platform. It aggregates msgs and forwards them to the consumer. Yet, this system has significant drawbacks. One is that high priority consumers might starve due to the lack of QoS in the event channel. The broker has no idea what is the deadline and time requirement of each consumer. So there might be possible priority inversion. So, TAO has added this feature to allow consumers to specify their requirement using connection parameters. The second issue is that broker will send all msgs it receives to all the consumers connected to it, which is surely not desirable. Not only does this kind of pattern waste bandwidth, but it also will increase CPU utilization on both broker and consumers side. It is rather natural to implement the event filtering mechanism. So, the way TAO implements this feature is really similar to the modern idea of publication and subscription. In other words, multiple topic channels can be created in the broker to facilitate the categorization. One thing that is pretty novel is that TAO also adds the logical AND & OR event to satisfy the consumers' requirements. So, in order to this, we might want add an extra layer of dependency between different topic channels so that the broker will not need to iterate through all the channels to find the right ones to publish every time. The third issue is that there is no support for periodic processing. TAO's RT Event Service allows consumers to specify event dependency timeouts so that consumers can meet their requirements. For the real-time scheduling system, I am basically agreed with his implementation. It is pretty natural to use the EFD policy to determine which one to schedule first. Moreover, the use of priority queue helps the system to achieve the lowest complexity as possible (log(n)). Among the different dispatchers, RTU dispatcher is the neatest as there is only one thread handling everything. Yet in some cases, the extra latency might pull down the whole system. On the other hand, the RT preemptive thread dispatching is the most optimal one to me. While there might be relatively large overhead due to context switching and high CPU utilizations, it leverages the OS thread to handle different priority requirement. As nowadays, the servers have even more cores and are able to support more threads, having the kernel threads dedicate to one priority can largely reduce the latency of high priority consumers.