Bottleneck of previous design:

1. Packets are processed one-by-one. No parallel processing.
2. Packet latency is dependent on output buffer size.
3. Primary-backup 2PC takes time, if we reduce buffer size→frequency of performing 2PC increases.
4. Consensus is performed per global state update to achieve strong consistency.
5. Input buffer size is infinite.
6. Stamping unit can be a single point of failure.-replicated stamping

Design Choices:

1. Multi-threaded implementation. Each thread handles k flows. Goal - to improve per packet latency
   1. What will be the advantages?
2. To tackle limited buffer size- explicit state migration during scaling and failover needs to be avoided.
   1. Cluster based approach. Each NF in a cluster shares states among clusters after each batch. This will avoid explicit state migration to an NF during scaling.
      1. What is the advantage?
      2. Load distribution among clusters
      3. Do we need back ups for each NF?
   2. Failover- flows can be redirected to one of the cluster members. But how cluster members will get the state update of the current batch the failed NF was processing? Packet duplication to all members (!)
3. Propose a load balancing algorithm among the cluster NFs and across clusters. Run experiments without and with this load-balancing scheme.