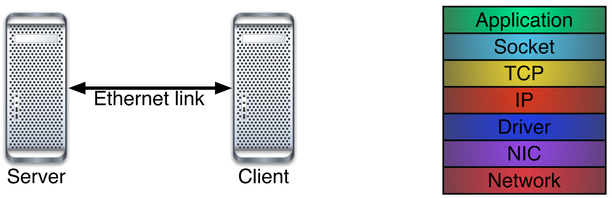
# Full-System Critical Path Analysis

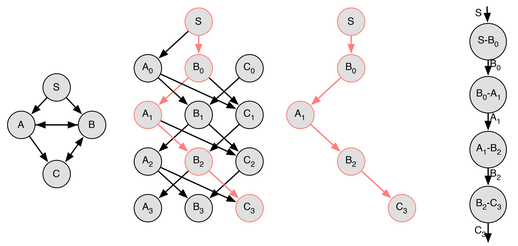
University of Michigan

Ali Saidi, Nathan Binkert, Steve Reinhardt, Trevor Mudge

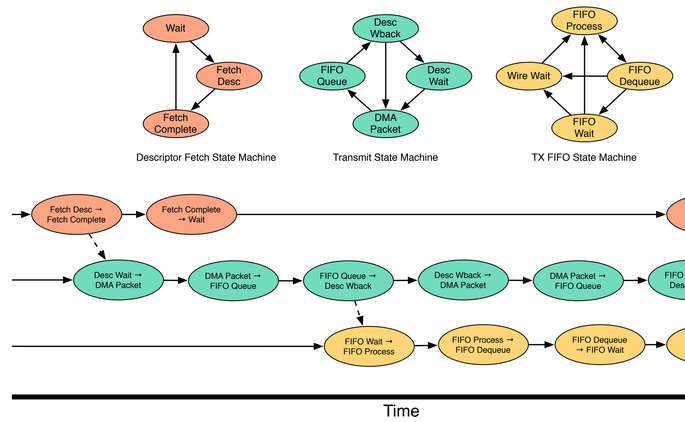
- Evaluating performance of systems difficult
  - End-to-end behavior is result of many interactions
    - Concurrently operating hardware and software components
    - When trying to improve performance, problem isn't clear
  - Locating the bottleneck component is difficult
    - Ad-hoc methods can be faulty
    - State-space exploration infeasible
- Particularly problematic in networking
  - Performances losses come from interactions between protocol, software, memory system, and NIC
  - End-to-end TCP bandwidth isn't as expected:
    - Range of possible reasons from spanning application, kernel TCP stack, driver, NIC, and network
    - Simply observing snapshot of state is insufficient



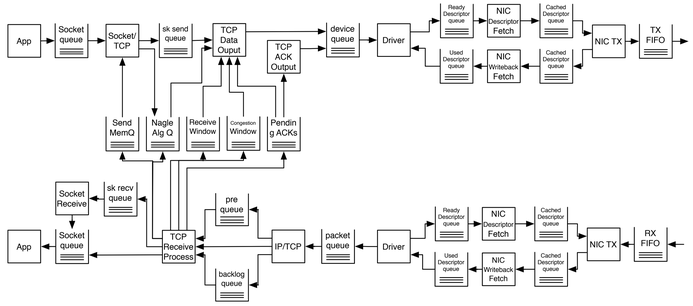

- Identify bottlenecks with Critical Path Analysis
  - Prerequisite is a dependence graph representing timing constraints
    - Simple for small system with few events
    - Becomes much harder for bigger systems
  - Algorithmically map of state machines to dependence graph
    - Execution of each state machine is converted to graph
    - Dependencies between state machines are added manually
  - Nodes represent a state change in the dependence graph
  - Edge weight represent time spent in a state



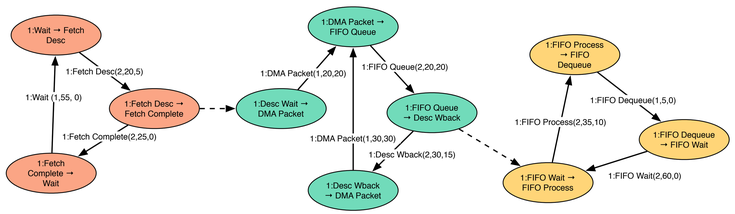

- State machines interact when one produces an output the other consumes
  - Can largely been seen as a series of queues
  - Act of removing an item from a queue sets up a dependence between the producer of items in queue and consumer
  - Queue push/pop/empty/full operations explicitly annotated
  - Edges between state machines weight is com latency
  - Reduces global dependence graph generation to description of local state machines and their local interactions




- Resulting graph is a fully connected DAG
  - Critical path can be found with standard graph analysis
  - Directly identifies performance bottleneck
  - Predict performance improvement when problems fixed
- Software state machines
  - Automatically convert functions (symbols) into states
  - User must manually mark pieces of code as belonging to a state machine
  - Can be done iteratively
- Iterative Construction
  - Complete, detailed decomposition not required
  - A little work can be done and then the analysis can drive future work
    - Missing dependencies when states are occupied for a very long time
    - Analysis program warns when interactions appear incorrect




- Visualizing the analysis
  - Can't visualize full graph (millions of nodes)
  - Compress the information in combination of state machines and bottleneck graph




- Loose Loops
  - Long loops in the critical path
  - Source of performance problems
  - Analysis can automatically identify them
  - Predict performance if the paths are broken
- Example output
  - TCP Stream between two systems with large link delay

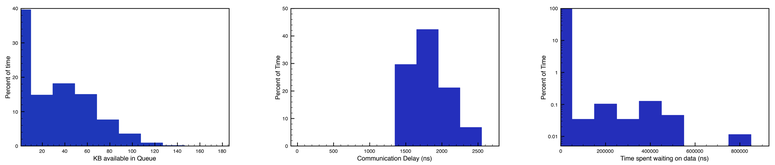
```
Most critical edges (time spent) on critical path 0 (12902788000):
-------------------------------------------------------------------
drivesys:RX Desc Wback:Queue__RX Used Desc->drivesys:E1000 RX:Dequeue__RX Used Desc     02.98 (00.00% waiting)
testsys:TcpSendMsg:__copy_user                                                         03.80 (10.24% waiting)
testsys:RX Desc Wback:Queue__RX Used Desc->testsys:E1000 RX:Dequeue__RX Used Desc      04.02 (00.00% waiting)
testsys:TcpRcvProcess:Queue__sndmemQ->testsys:TcpSendMsg:Dequeue__sndmemQ              10.37 (00.00% waiting)
testsys:TcpSendMsg:tcp_push_one                                                        14.45 (99.46% waiting)
drivesys:TXQ:Queue__WireQ->testsys:RXQ:Dequeue__WireQ                                  15.50 (00.00% waiting)
testsys:TXQ:Queue__WireQ->drivesys:RXQ:Dequeue__WireQ                                  19.42 (00.00% waiting)
drivesys:TcpRcvProcess:Queue__ackQ->drivesys:TcpAck:Dequeue__ackQ                      24.09 (00.00% waiting)

Most critical edges (loose loops) on critical path:
---------------------------------------------------
3: testsys:TcpRcvProcess:Queue__sndmemQ -> testsys:TcpSendMsg:Dequeue__sndmemQ

Most critical edges (time spent) on critical path 1 (12600624500):
-------------------------------------------------------------------
testsys:TcpSendMsg:tcp_push_one                                                        03.18 (99.53% waiting)
testsys:RX Desc Wback:Queue__RX Used Desc->testsys:E1000 RX:Dequeue__RX Used Desc      04.30 (00.00% waiting)
drivesys:RX Desc Wback:Queue__RX Used Desc->drivesys:E1000 RX:Dequeue__RX Used Desc    04.49 (00.00% waiting)
drivesys:TcpRcvProcess:Queue__ackQ->drivesys:TcpAck:Dequeue__ackQ                      24.70 (00.00% waiting)
drivesys:TXQ:Queue__WireQ->testsys:RXQ:Dequeue__WireQ                                  27.78 (00.00% waiting)
testsys:TXQ:Queue__WireQ->drivesys:RXQ:Dequeue__WireQ                                  31.82 (00.00% waiting)

Most critical edges (loose loops) on critical path:
---------------------------------------------------
6: testsys:TcpRcvProcess:Queue__wndQ -> testsys:TcpOutput:Peek__wndQ
1: testsys:TcpRcvProcess:Queue__cwndQ -> testsys:TcpOutput:Peek__cwndQ

Most critical edges (time spent) on critical path 2 (6909148000):
-------------------------------------------------------------------
testsys:TcpSendMsg:kmem_cache_alloc                                                    01.27 (00.00% waiting)
testsys:TcpSendMsg:alloc_skb                                                           01.75 (11.26% waiting)
drivesys:RX Desc Wback:Queue__RX Used Desc->drivesys:E1000 RX:Dequeue__RX Used Desc    01.81 (00.00% waiting)
testsys:TcpSendMsg:__kmalloc                                                           01.88 (21.72% waiting)
testsys:TcpSendMsg:tcp_sendmsg                                                         02.67 (12.15% waiting)
testsys:TcpSendMsg:release_sock                                                        05.96 (99.06% waiting)
testsys:TXQ:Queue__WireQ->drivesys:RXQ:Dequeue__WireQ                                  07.25 (00.00% waiting)
testsys:TcpSendMsg:__copy_user                                                         16.87 (14.34% waiting)
testsys:TcpSendMsg:tcp_push_one                                                        57.71 (99.48% waiting)
```




- Future Work
  - Analysis currently limited to single streams
  - Apply techniques to larger workloads