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 weights 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

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