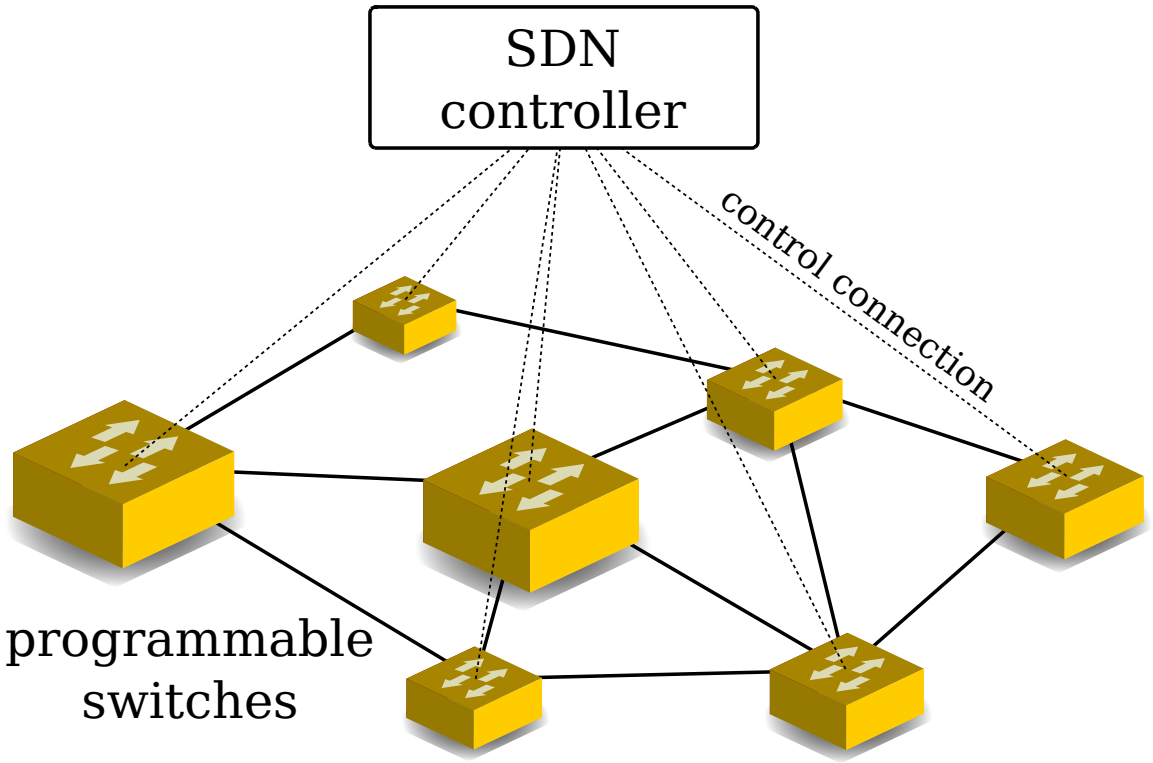
Taming SDN Controllers in Heterogeneous Hardware Environments

Zdravko Bozakov Amr Rizk

Leibniz Universität Hannover
Fakultät für Elektrotechnik und Informatik
Institut für Kommunikationstechnik

EWSDN 2013, 10-11 October, Berlin

Software Defined Network abstraction



- Centralized controller
- Collection of programmable forwarding devices
- Communication over well-defined API (e.g. OpenFlow)

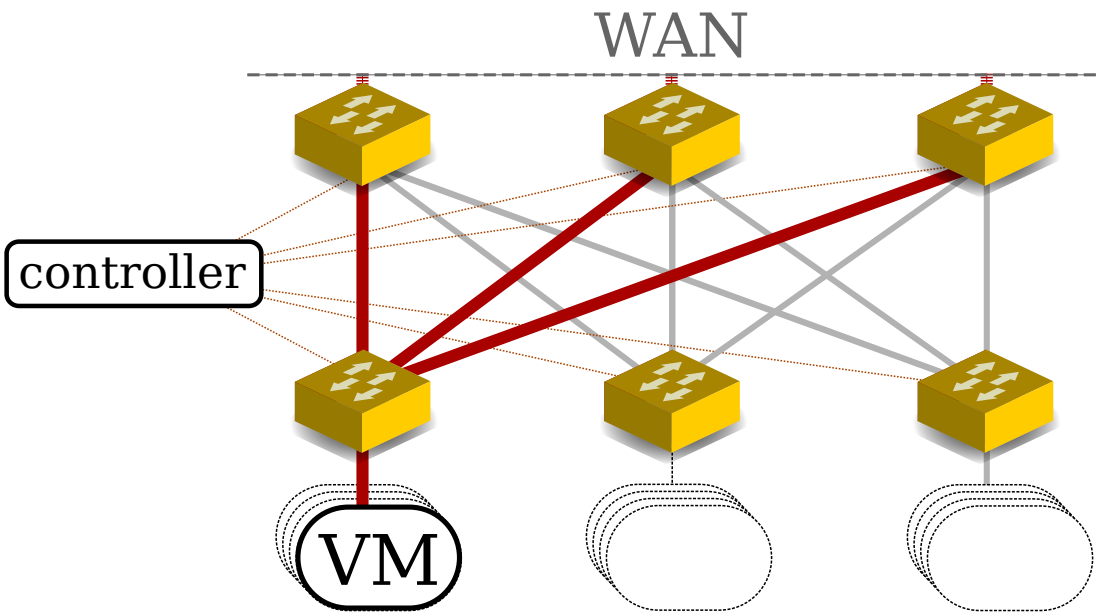
Heterogeneity of forwarding devices is an inherent property of Software Defined Networks

How does this effect the responsiveness of SDN applications?

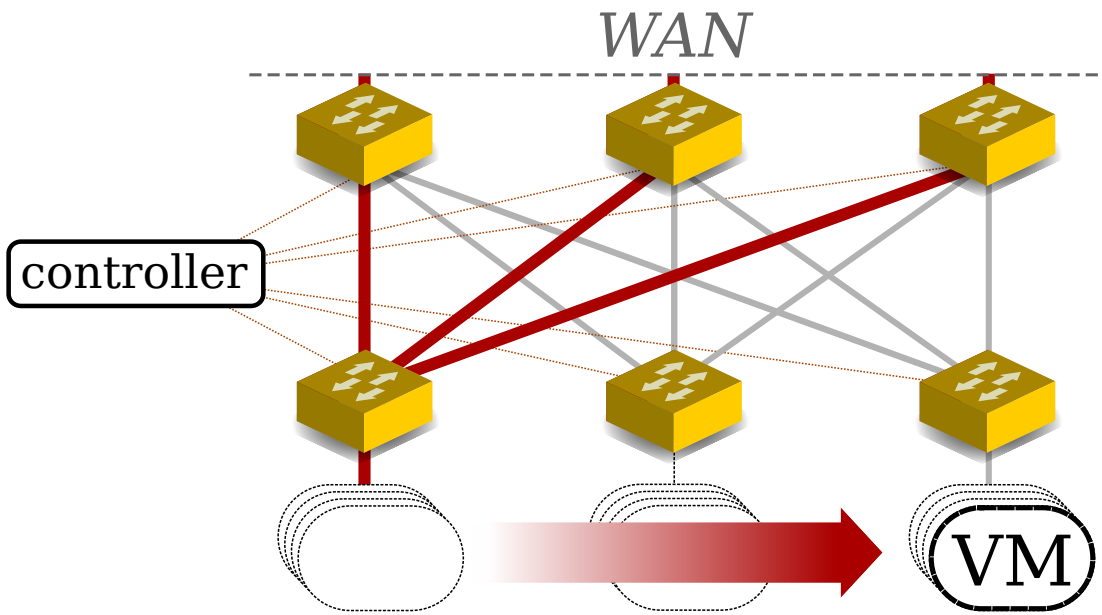
Heterogeneity of forwarding devices is an inherent property of Software Defined Networks

How does this effect the responsiveness of SDN applications?

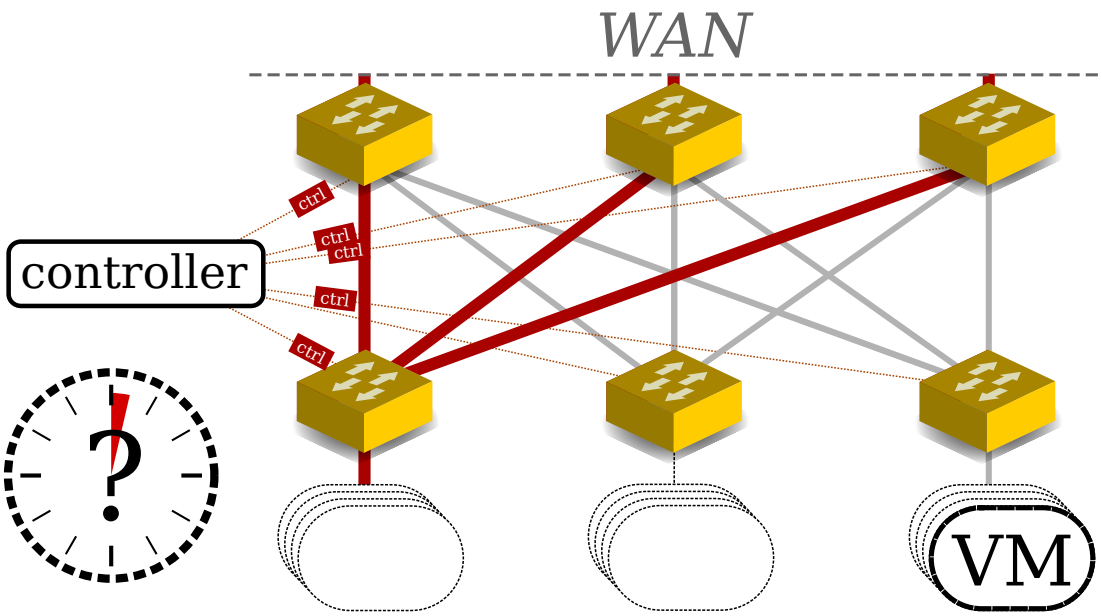
Varying processing capability of control logic in switches yields unpredictable delays for SDN applications



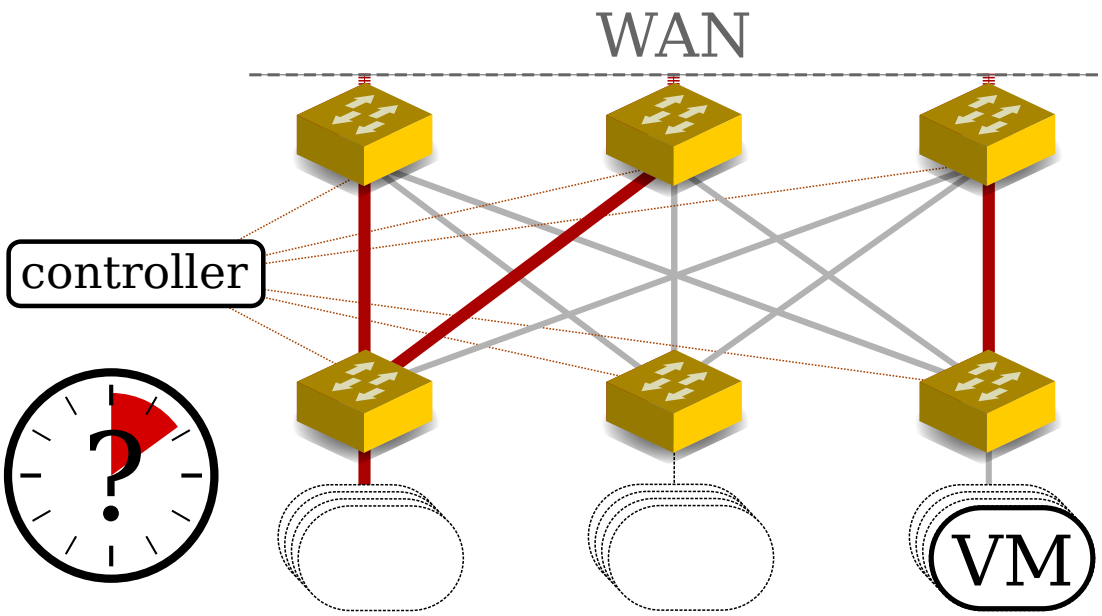
Varying processing capability of control logic in switches yields unpredictable delays for SDN applications



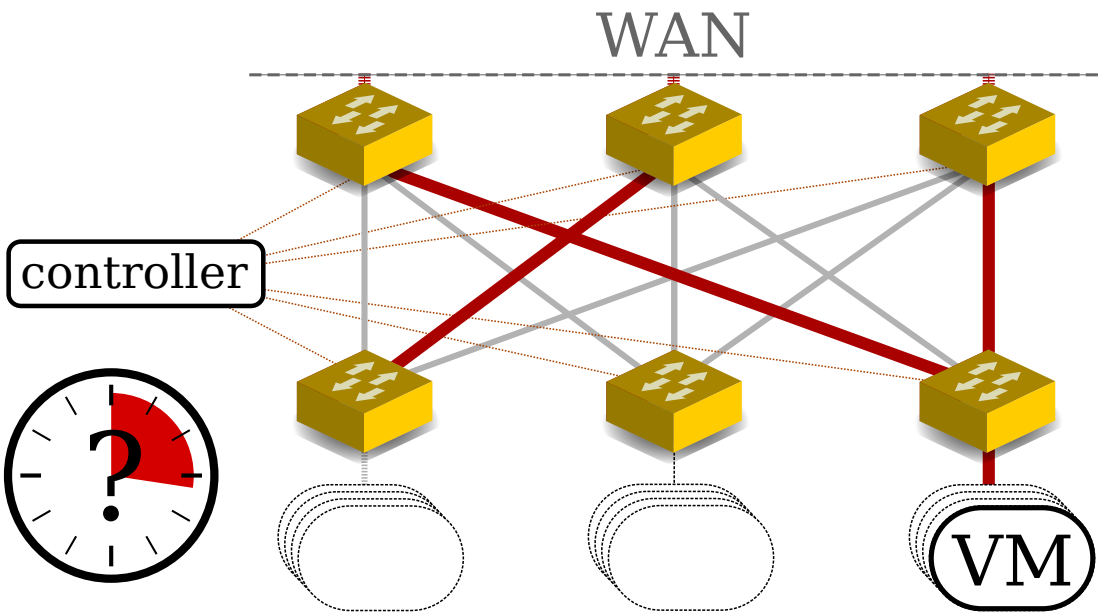
Varying processing capability of control logic in switches yields unpredictable delays for SDN applications



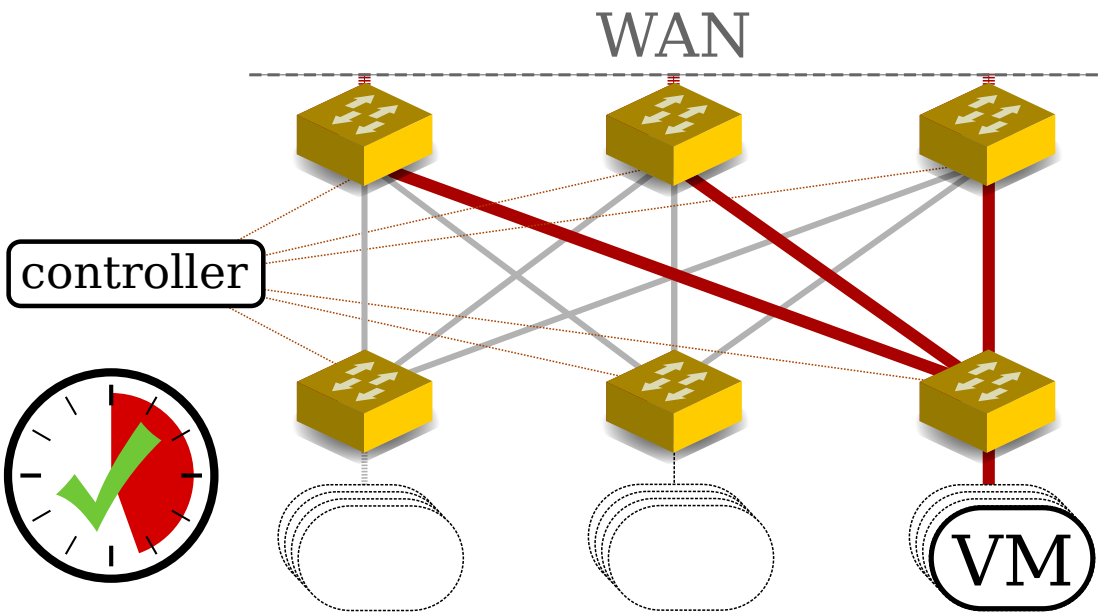
Varying processing capability of control logic in switches yields unpredictable delays for SDN applications



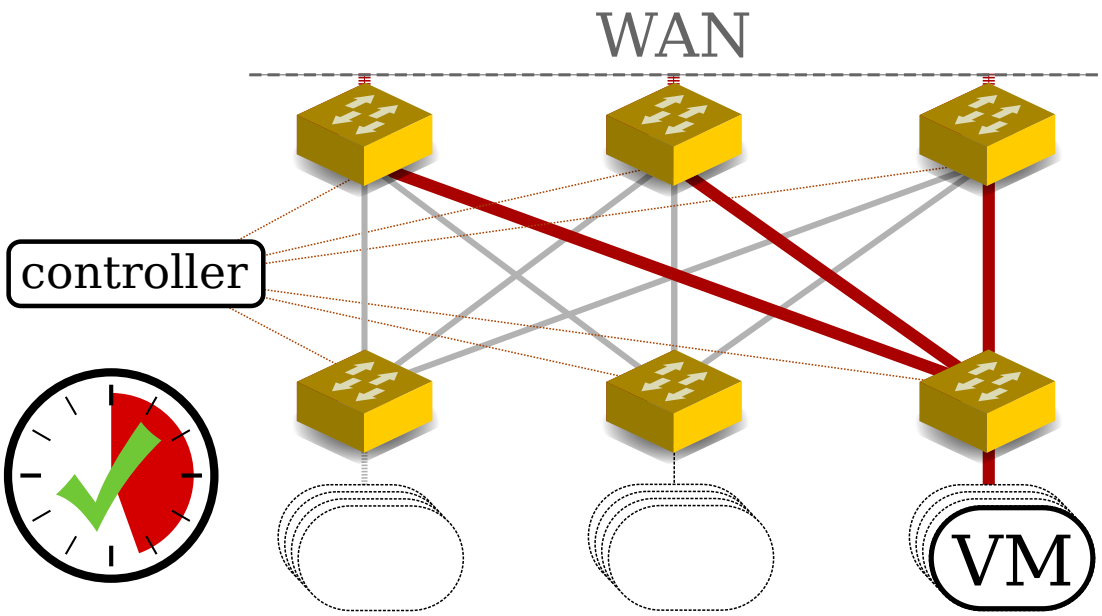
Varying processing capability of control logic in switches yields unpredictable delays for SDN applications



Varying processing capability of control logic in switches yields unpredictable delays for SDN applications



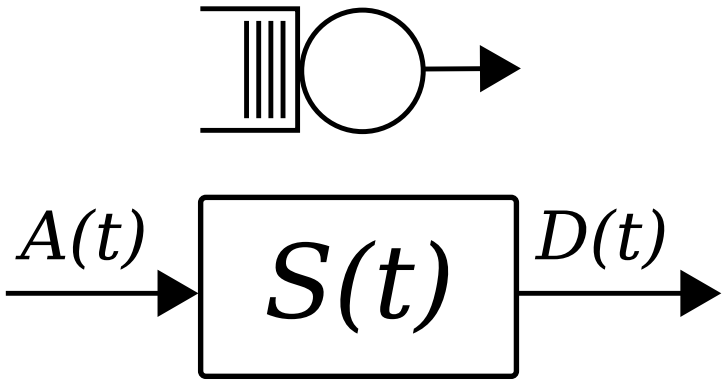
Varying processing capability of control logic in switches yields unpredictable delays for SDN applications

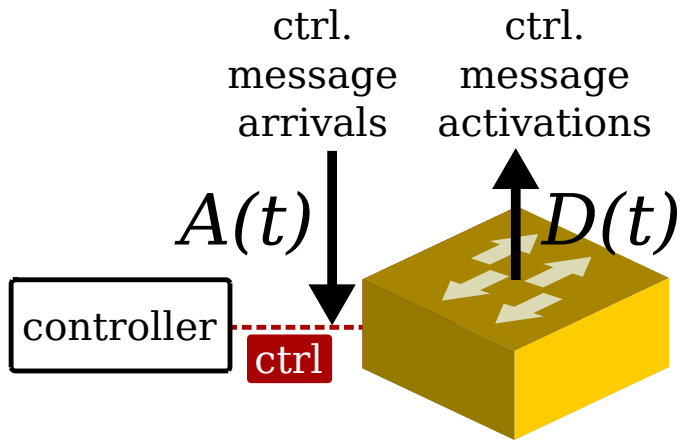


Main causes for flow installation latencies at a switch

- number of messages queued for processing (load dependent)
- rate of processing control messages (switch dependent)

We model the control message processing mechanism of SDN switches as a queueing model



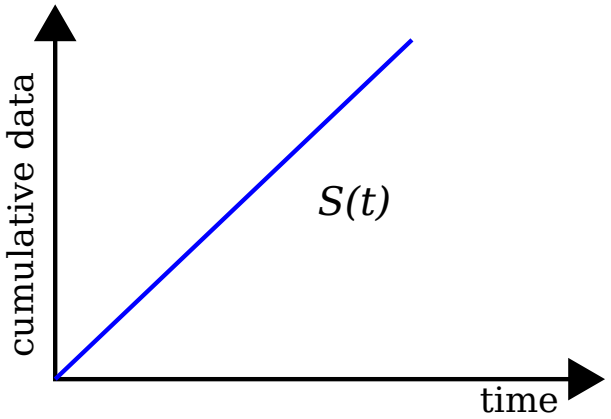


$A(t)$ control message arrivals at switch

$D(t)$ control message activations

$S(t)$ amount of messages processed over different time intervals

We model the control message processing mechanism of SDN switches as a queueing model

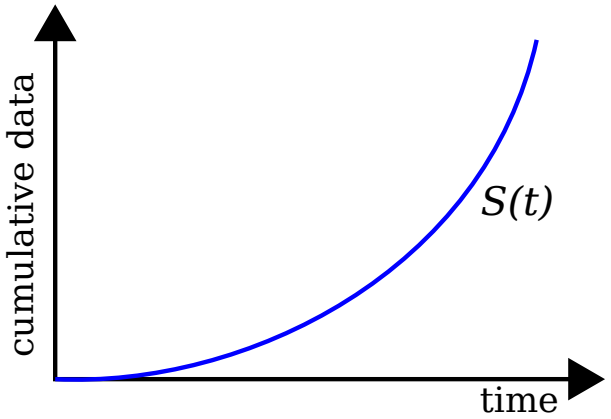


$A(t)$ control message arrivals at switch

$D(t)$ control message activations

$S(t)$ amount of messages processed over different time intervals

We model the control message processing mechanism of SDN switches as a queueing model

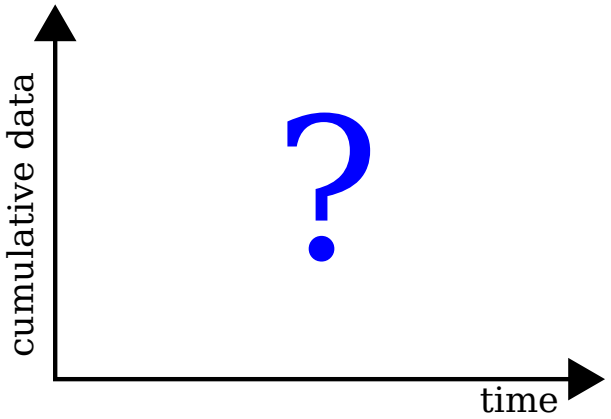


$A(t)$ control message arrivals at switch

$D(t)$ control message activations

$S(t)$ amount of messages processed over different time intervals

We model the control message processing mechanism of SDN switches as a queueing model



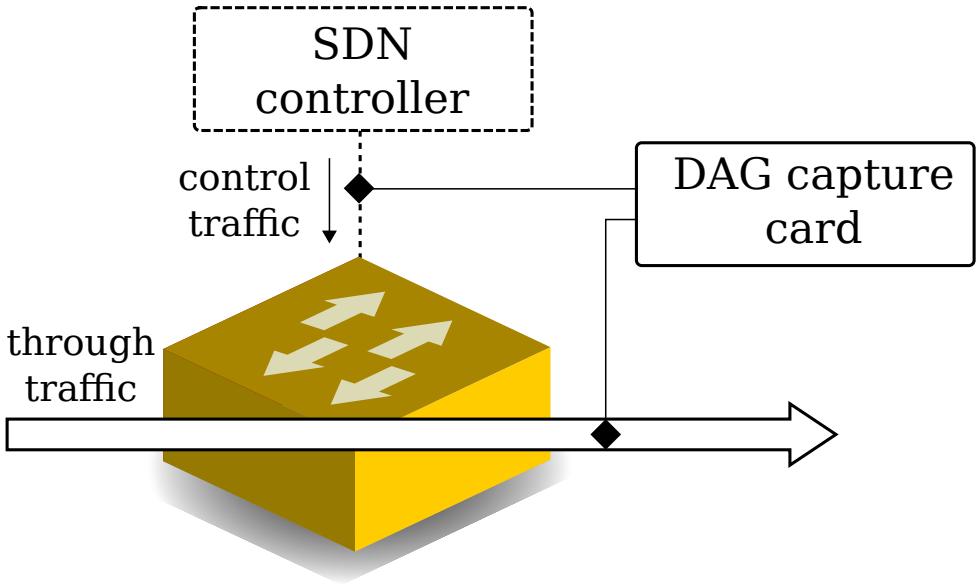
$A(t)$ control message arrivals at switch

$D(t)$ control message activations

$S(t)$ amount of messages processed over different time intervals

Experimental setup for service curve estimation:

- forward constant (maximal) rate through traffic
- inject a burst of control messages into switch with *flowmod* action (or other)
- capture through traffic and evaluate times between modified header fields

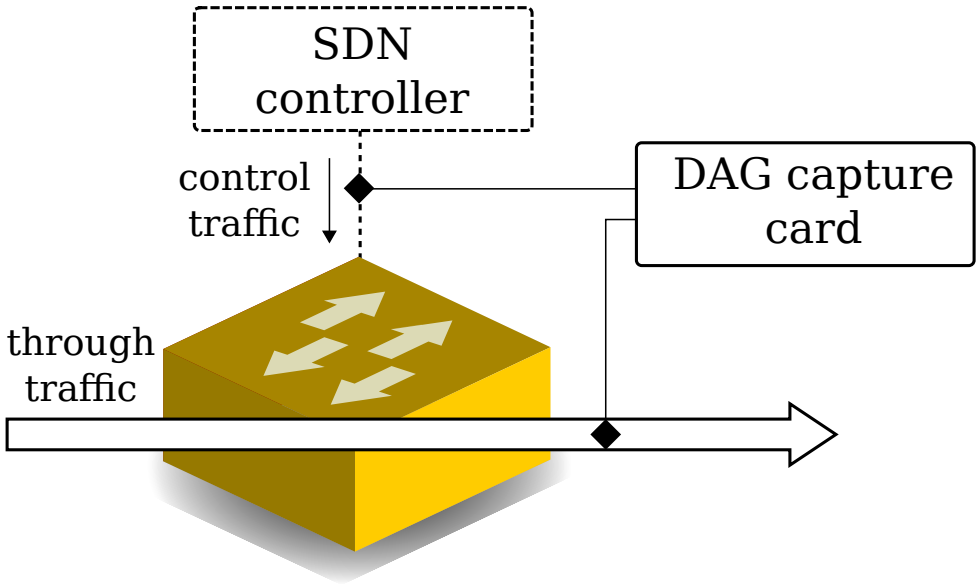


Evaluated two switches implementing OpenFlow 1.0

- OpenVSwitch on quad core Xeon server
- pica8 48x1GB port switch

Experimental setup for service curve estimation:

- forward constant (maximal) rate through traffic
- inject a burst of control messages into switch with *flowmod* action (or other)
- capture through traffic and evaluate times between modified header fields

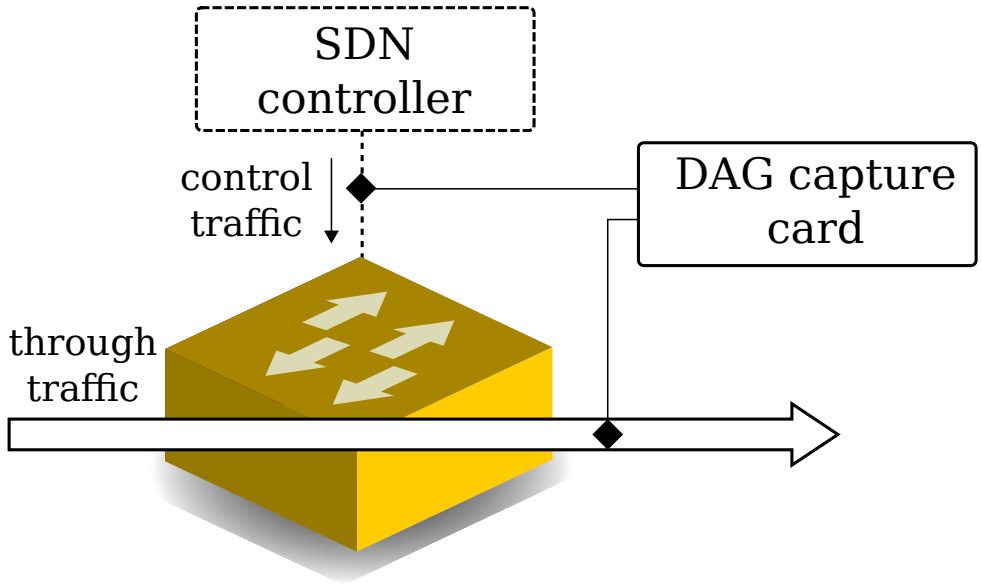


Evaluated two switches implementing OpenFlow 1.0

- OpenVSwitch on quad core Xeon server
- pica8 48x1GB port switch

Experimental setup for service curve estimation:

- forward constant (maximal) rate through traffic
- inject a burst of control messages into switch with *flowmod* action (or other)
- capture through traffic and evaluate times between modified header fields

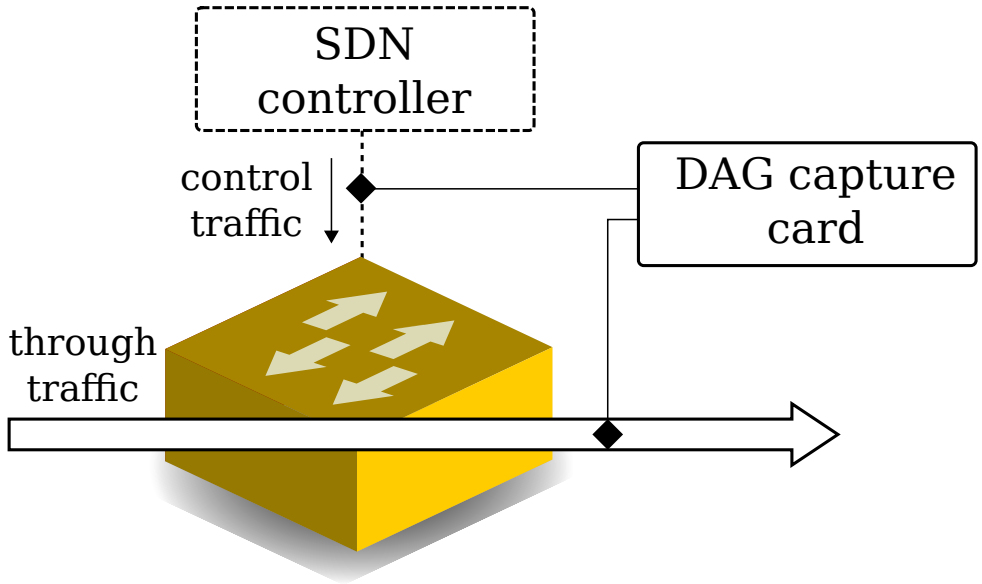


Evaluated two switches implementing OpenFlow 1.0

- OpenVSwitch on quad core Xeon server
- pica8 48x1GB port switch

Experimental setup for service curve estimation:

- forward constant (maximal) rate through traffic
- inject a burst of control messages into switch with *flowmod* action (or other)
- capture through traffic and evaluate times between modified header fields

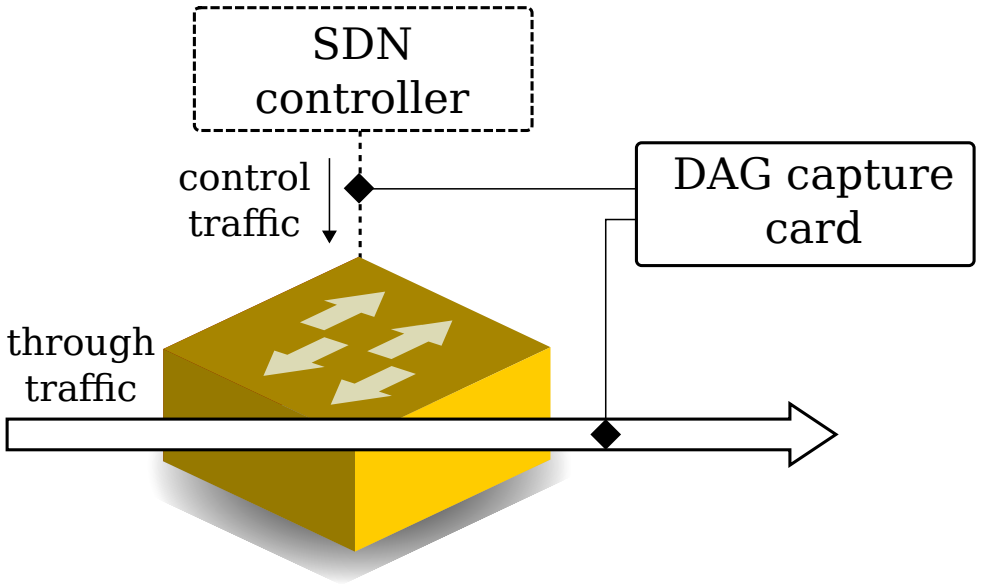


Evaluated two switches implementing OpenFlow 1.0

- OpenVSwitch on quad core Xeon server
- pica8 48x1GB port switch

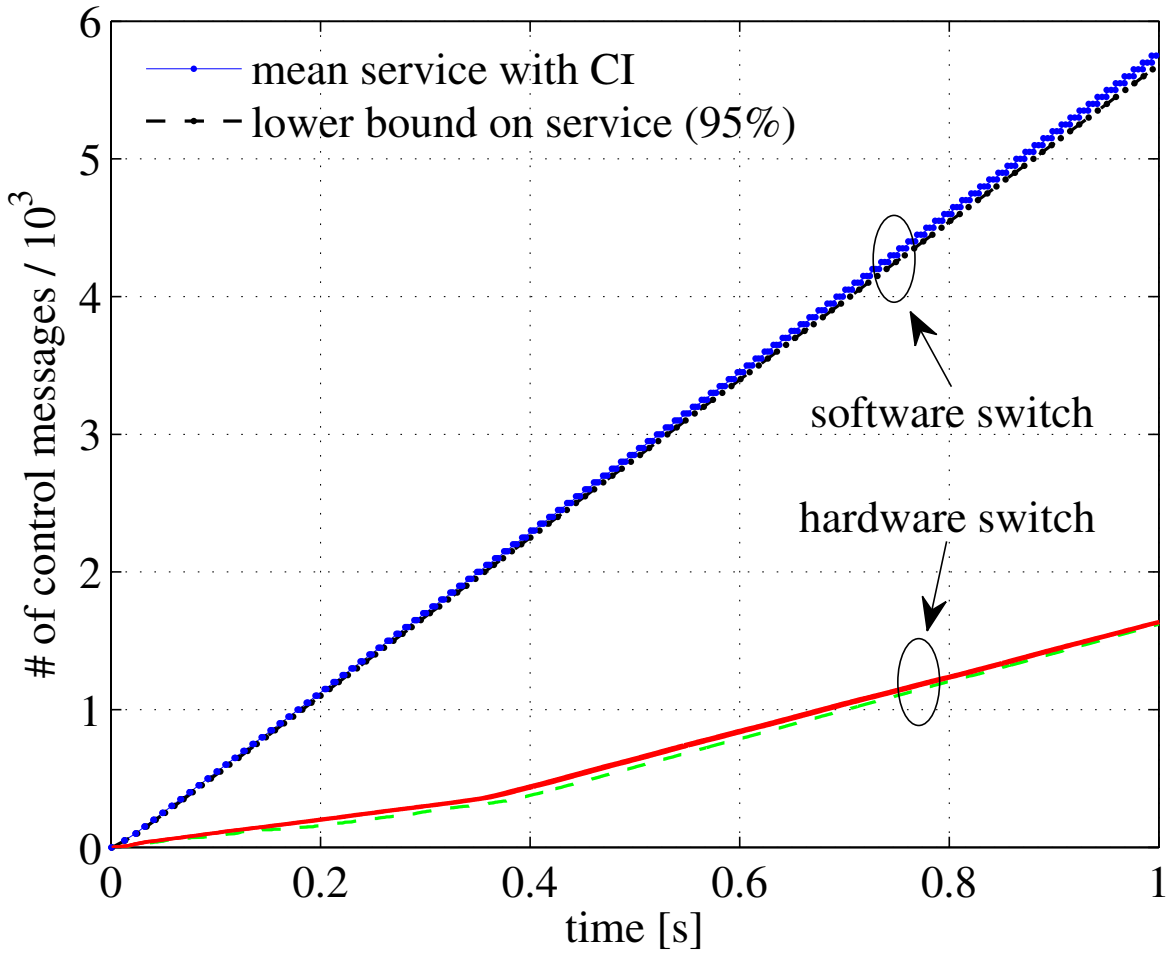
Experimental setup for service curve estimation:

- forward constant (maximal) rate through traffic
- inject a burst of control messages into switch with *flowmod* action (or other)
- capture through traffic and evaluate times between modified header fields



Evaluated two switches implementing OpenFlow 1.0

- OpenVSwitch on quad core Xeon server
- pica8 48x1GB port switch

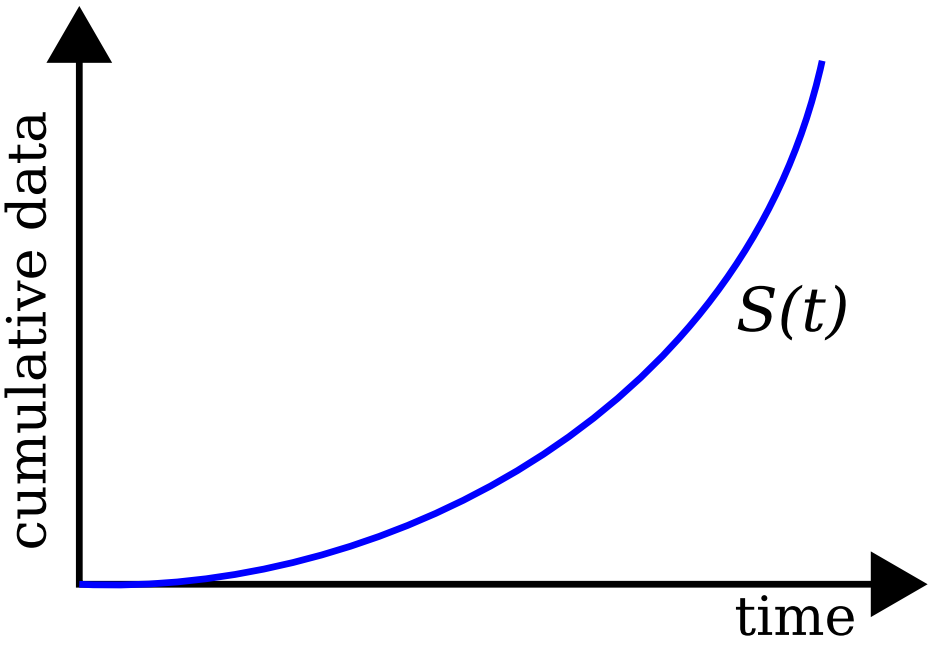

mean service with CI
lower bound on service (95%)
software switch
hardware switch
# of control messages / 10^3
time [s]
0
1
2
3
4
5
6
0
0.2
0.4
0.6
0.8
1

How can SDN applications seamlessly benefit from information about switch capabilities?

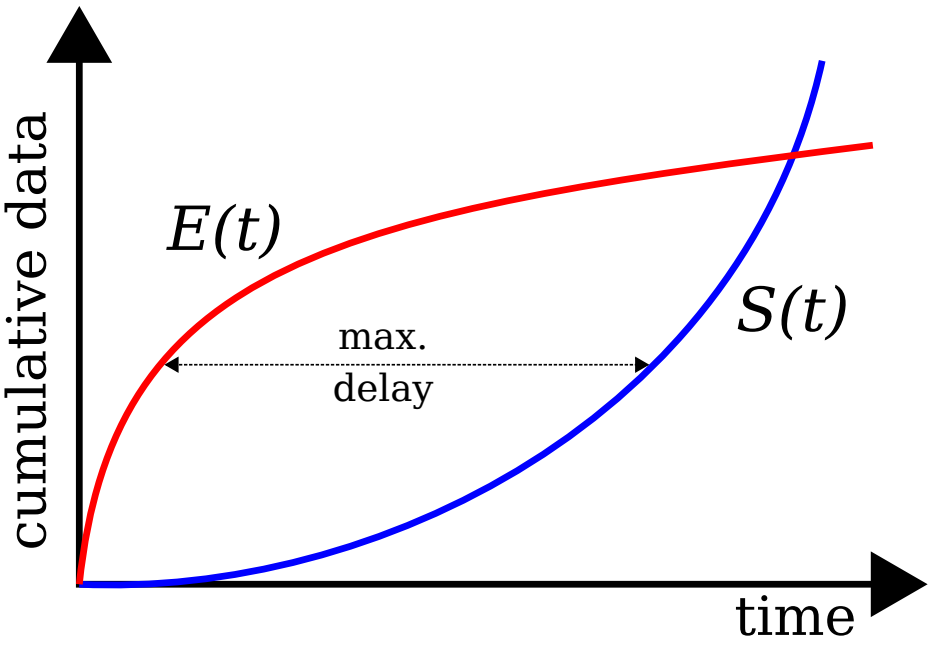
Current state of OpenFlow

- Rate limiting of control messages offered by some controller frameworks
- OpenFlow barrier messages enable coarse application control
- Applications cannot be tuned to heterogeneous switch capabilities
- No mechanism for estimating the maximum time required for a flow to become active at a switch.

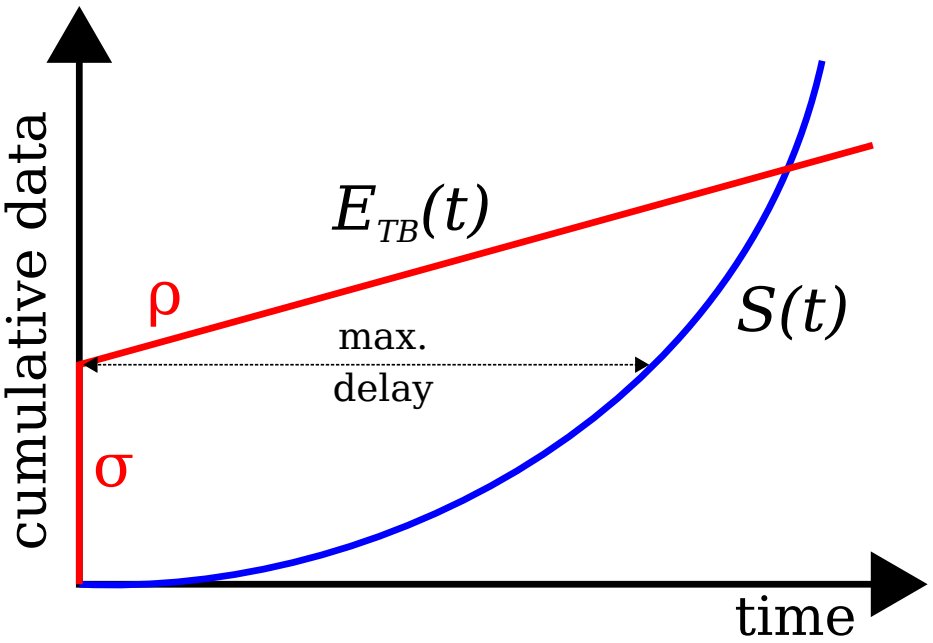
Given bounds on the arrival of control messages, network calculus model enables calculation of maximum delay bounds



Given bounds on the arrival of control messages, network calculus model enables calculation of maximum delay bounds



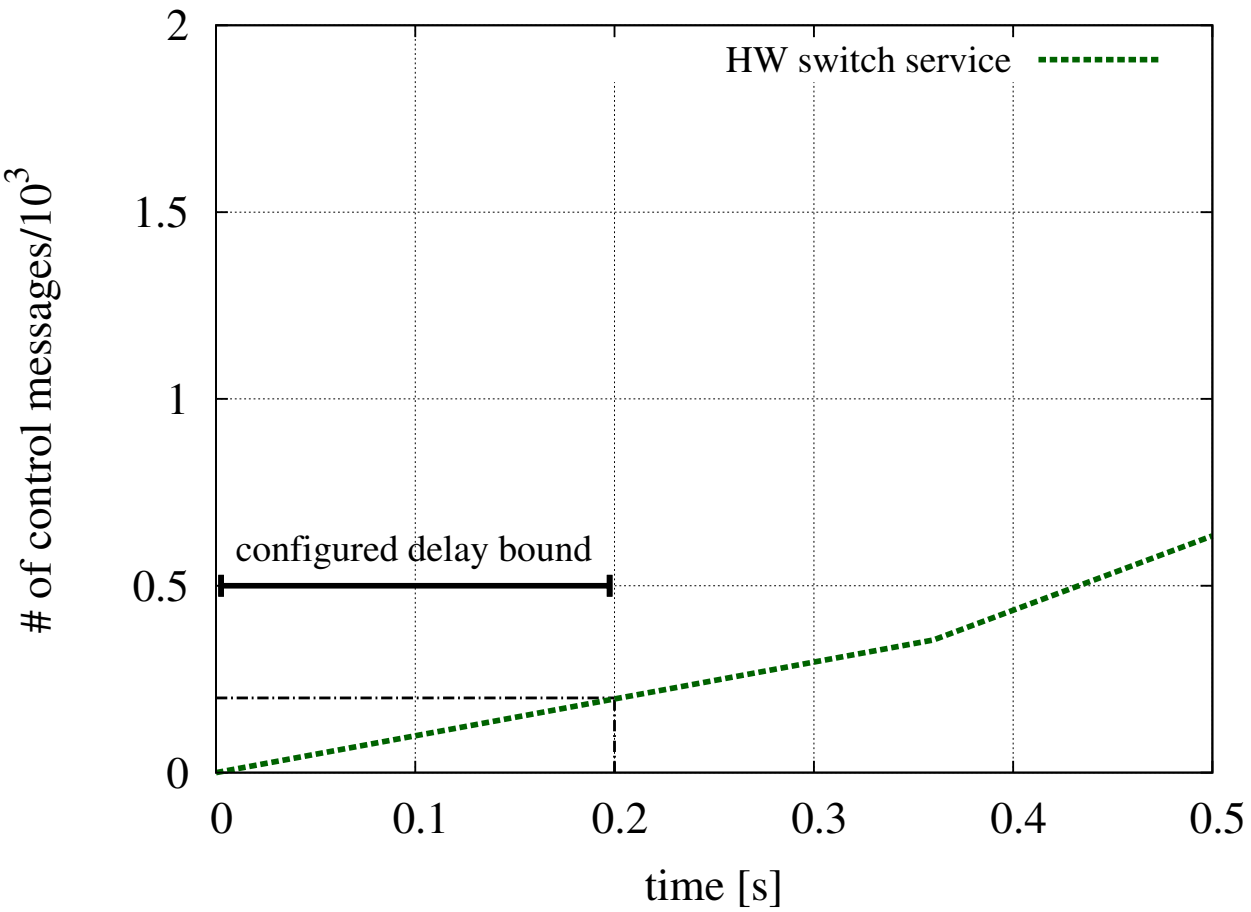
Given bounds on the arrival of control messages, network calculus model enables calculation of maximum delay bounds



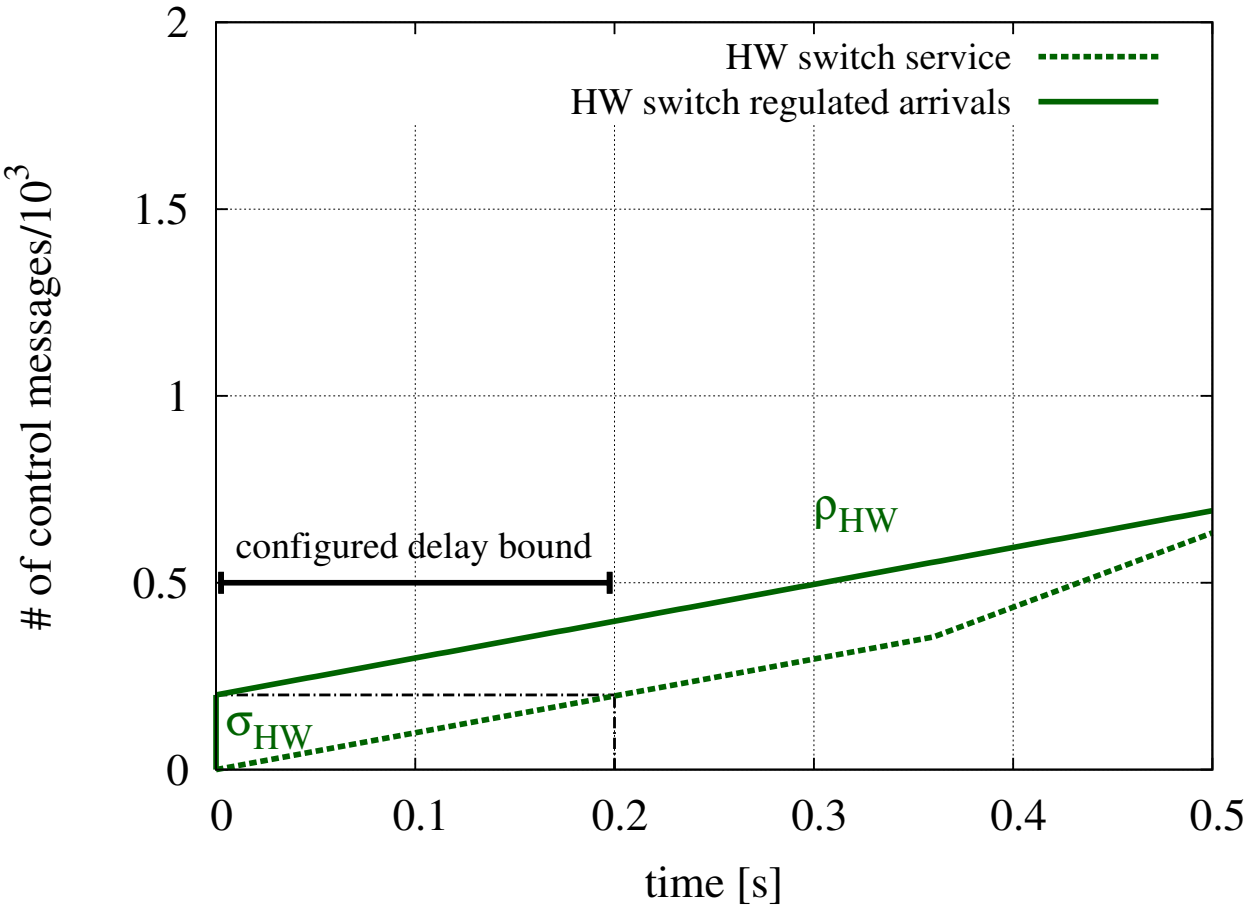
Proposed approach:

- extend interface of controller framework with *token bucket* regulator
- parametrize regulator for each type of deployed switch such that a specific delay bound holds

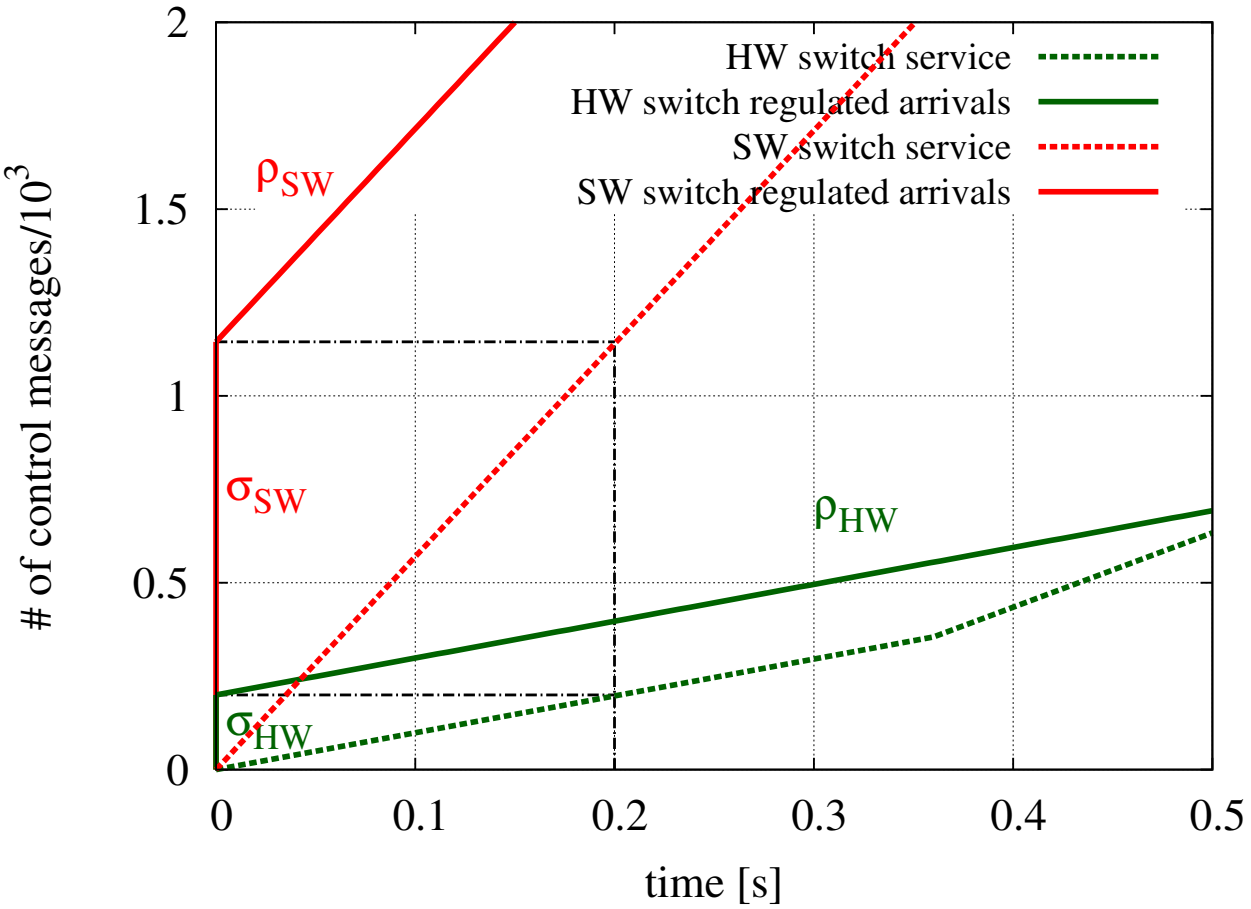
Example: configure maximum delay as 0.2s



Example: configure maximum delay as 0.2s



Example: configure maximum delay as 0.2s



Benefits of our approach

- Enables SDN applications to gauge flow instantiation time:
  - if control message is accepted by interface it will become active after a predefined maximum time
  - adapt message generation rate by querying processing rate and currently available number of tokens for each connected switch
- TB mechanism allows us to reserve part of switch service for high priority control messages
- Simple extension which does not alter current SDN architecture
- Service curve parameters may be stored in database or exchanged as part of control connection handshake (*feature_response*)

Benefits of our approach

- Enables SDN applications to gauge flow instantiation time:
  - if control message is accepted by interface it will become active after a predefined maximum time
  - adapt message generation rate by querying processing rate and currently available number of tokens for each connected switch
- TB mechanism allows us to reserve part of switch service for high priority control messages
- Simple extension which does not alter current SDN architecture
- Service curve parameters may be stored in database or exchanged as part of control connection handshake (*feature_response*)

Benefits of our approach

- Enables SDN applications to gauge flow instantiation time:
  - if control message is accepted by interface it will become active after a predefined maximum time
  - adapt message generation rate by querying processing rate and currently available number of tokens for each connected switch
- TB mechanism allows us to reserve part of switch service for high priority control messages
- Simple extension which does not alter current SDN architecture
- Service curve parameters may be stored in database or exchanged as part of control connection handshake (*feature_response*)

Benefits of our approach

- Enables SDN applications to gauge flow instantiation time:
  - if control message is accepted by interface it will become active after a predefined maximum time
  - adapt message generation rate by querying processing rate and currently available number of tokens for each connected switch
- TB mechanism allows us to reserve part of switch service for high priority control messages
- Simple extension which does not alter current SDN architecture
- Service curve parameters may be stored in database or exchanged as part of control connection handshake (*feature_response*)

Contributions

- We showed that device heterogeneity may lead to unpredictable behaviour in SDN applications and must be considered during design phase of increasingly complex SDN applications
- Outlined a model and a measurement approach to characterize control message processing capabilities of SDN switches
- Proposed an unintrusive controller framework mechanism enabling applications to consider substrate capabilities

Our work is a starting point for a number of research directions

- Extend to distributed controller frameworks
- What are the effects of higher abstraction layers on network responsiveness
- How should SDN applications be designed?

Contributions

- We showed that device heterogeneity may lead to unpredictable behaviour in SDN applications and must be considered during design phase of increasingly complex SDN applications
- Outlined a model and a measurement approach to characterize control message processing capabilities of SDN switches
- Proposed an unintrusive controller framework mechanism enabling applications to consider substrate capabilities

Our work is a starting point for a number of research directions

- Extend to distributed controller frameworks
- What are the effects of higher abstraction layers on network responsiveness
- How should SDN applications be designed?