NetQoPE: A Middleware-based Netowork QoS Provisioning Engine for Distributed Real-time and Embedded Systems

Jaiganesh Balasubramanian

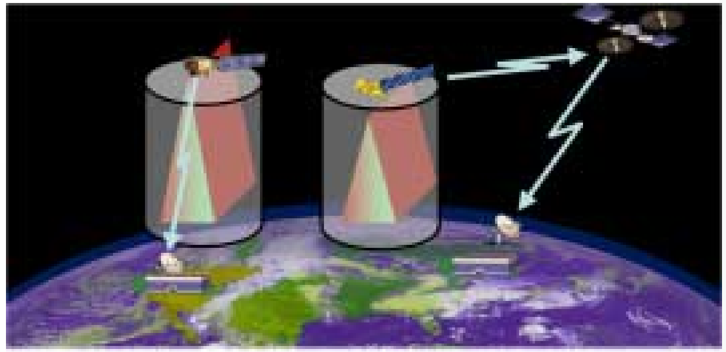
jai@dre.vanderbilt.edu

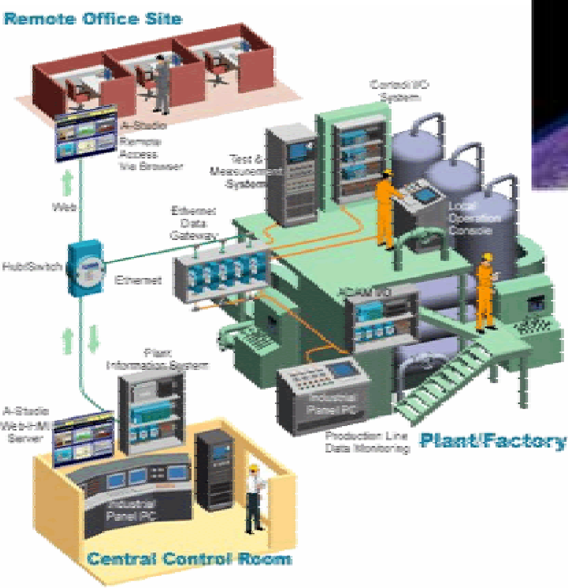
Work done in collaboration with
Sumant Tambe, Aniruddha Gokhale & Douglas C. Schmidt (Vanderbilt)
Shrirang Gadgil, Frederic Porter & Dasarathy Balakrishnan (Telcordia)

ISIS, Dept. of EECS
Vanderbilt University
Nashville, Tennessee

www.dre.vanderbilt.edu

Distributed Real-time & Embedded (DRE) Systems

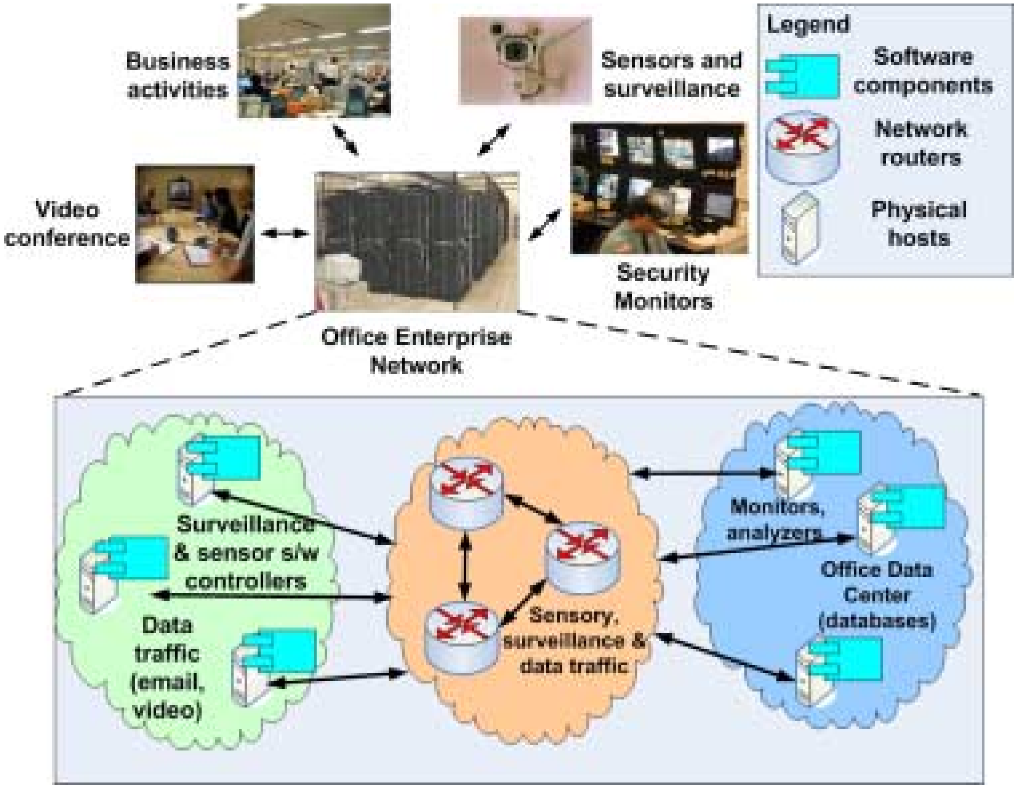




- Network-centric and large-scale "systems of systems"
  - e.g., industrial automation, emergency response
- Multiple end-to-end application flows that require various QoS properties
  - e.g., CPU, memory, network QoS
- Integrated solutions with network QoS mechanisms
  - e.g., Differentiated services (DiffServ), Integrated services (IntServ), Multi-protocol Layer Switching (MPLS)

DRE systems developed via robust and reliable system composition and integration of services and applications

Case Study: Modern Office Environment

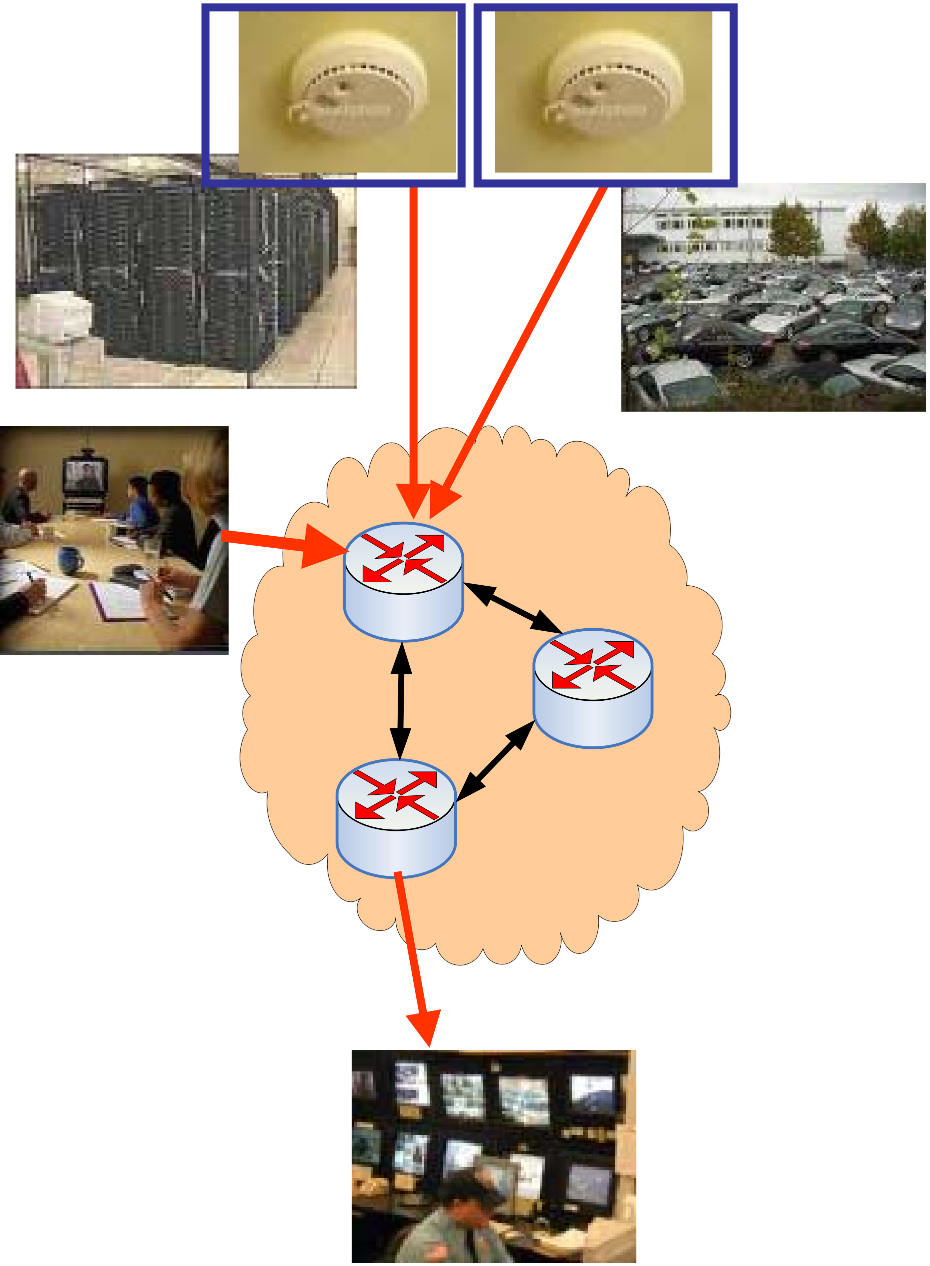


- Office traffic operates over IP networks & Fast ethernets
- Multiple application flows:
  - Email
  - Videoconferencing
  - Sensory (e.g., fire alarms)
- Differing network QoS requirements
  - Fire alarm – highest priority
  - Surveillance – multimedia
  - Temperature sensing – best effort
- QoS provisioned using DiffServ

Network QoS Provisioning Steps

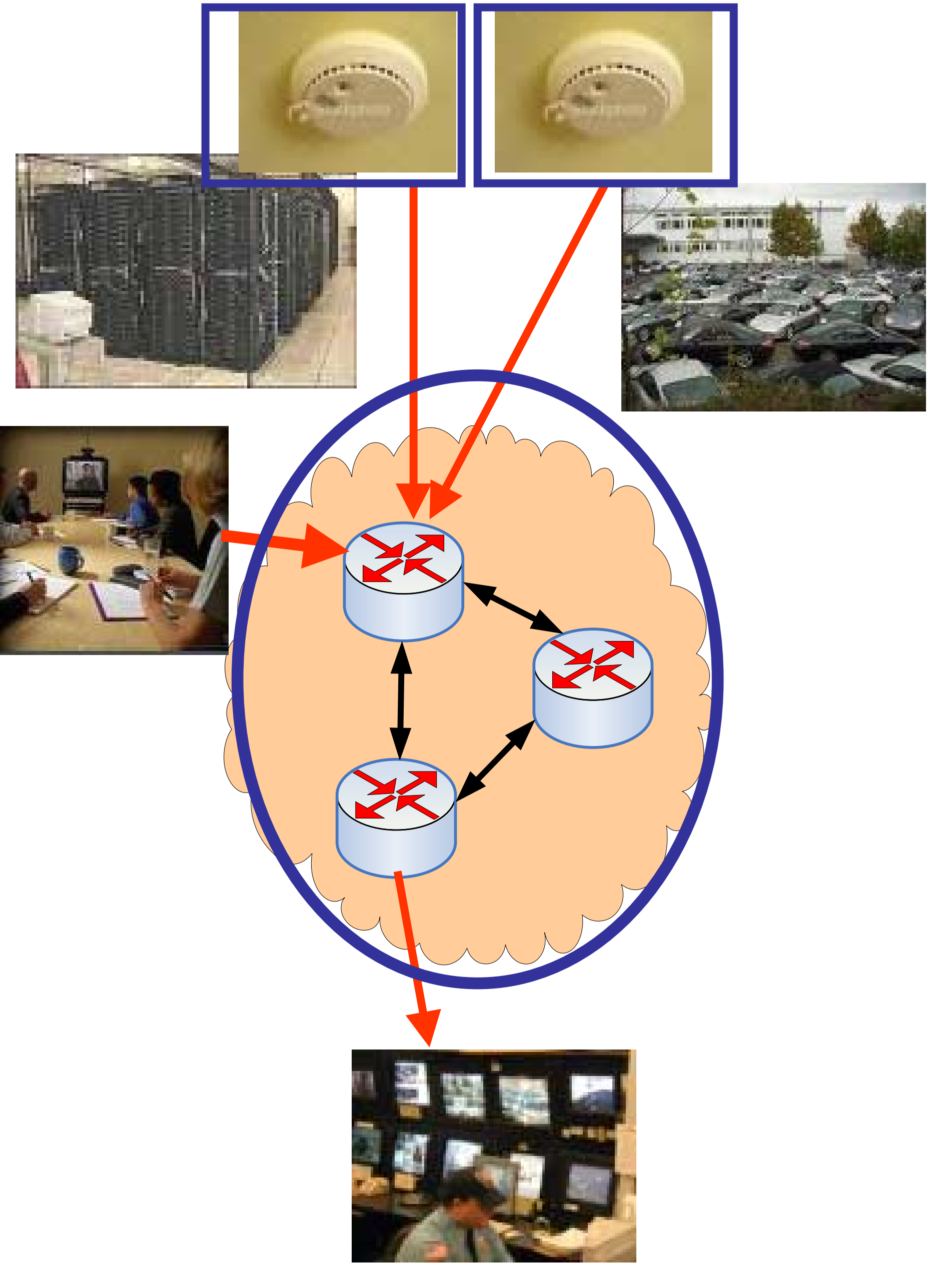
1. Specify network QoS requirements for each application flow
2. Allocate network-level resources and DiffServ Code Points (DSCP) for every application flow joining two end points
3. Mark outgoing packet with the right DSCP values

Challenge 1: QoS Requirements Specification



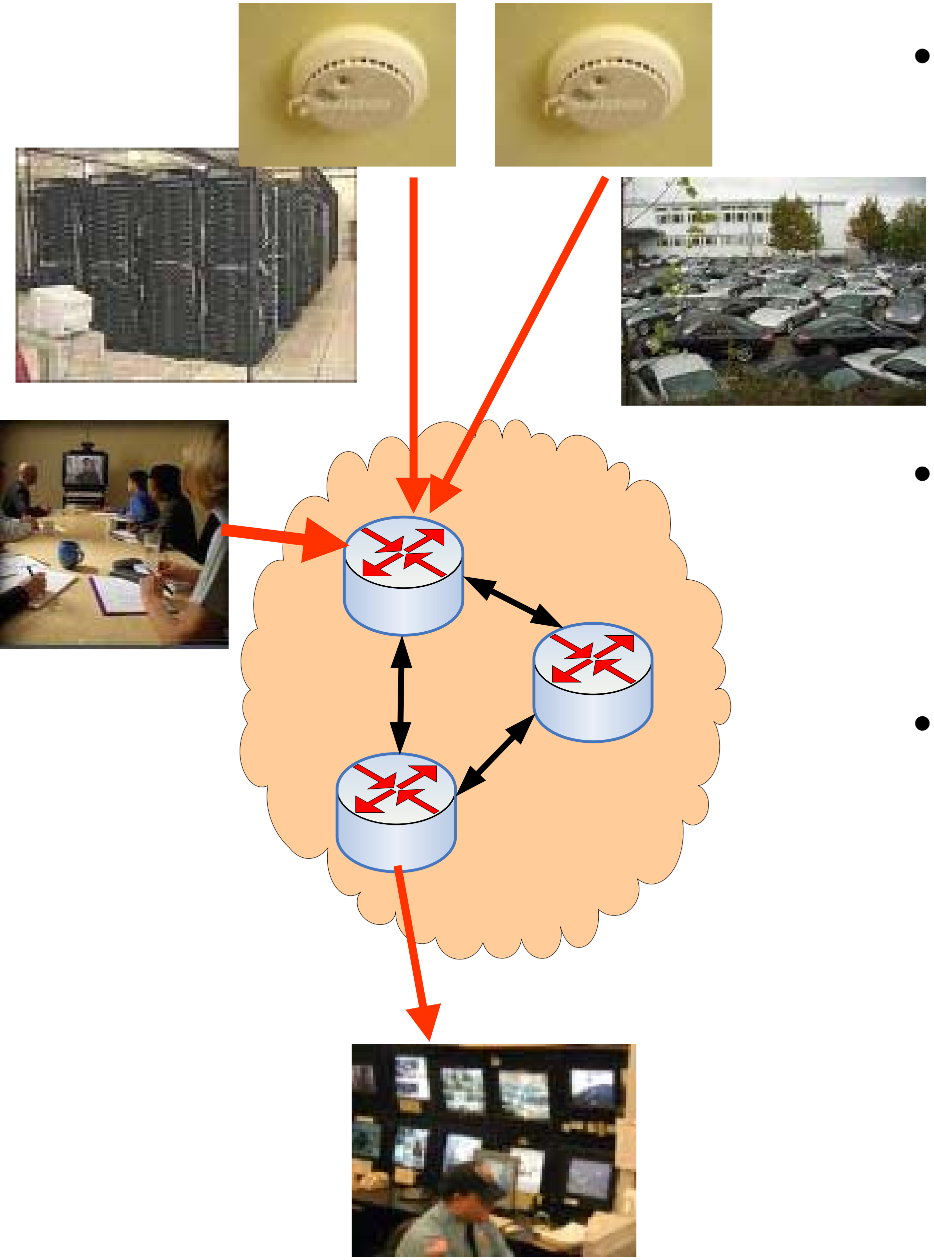
- Specify required levels of service
  - High priority or High Reliability
  - 10 Mbps forward and reverse bandwidth
- Reusable software controllers deploy the same application code
  - Fire sensor deployed in the parking lot as well as server room
  - Different network QoS requirements based on importance levels
- Problem
  - manual or programmatic specification of network QoS requirements
    - tedious
    - error-prone
    - non-scalable

Challenge 2: Network Resource Allocation



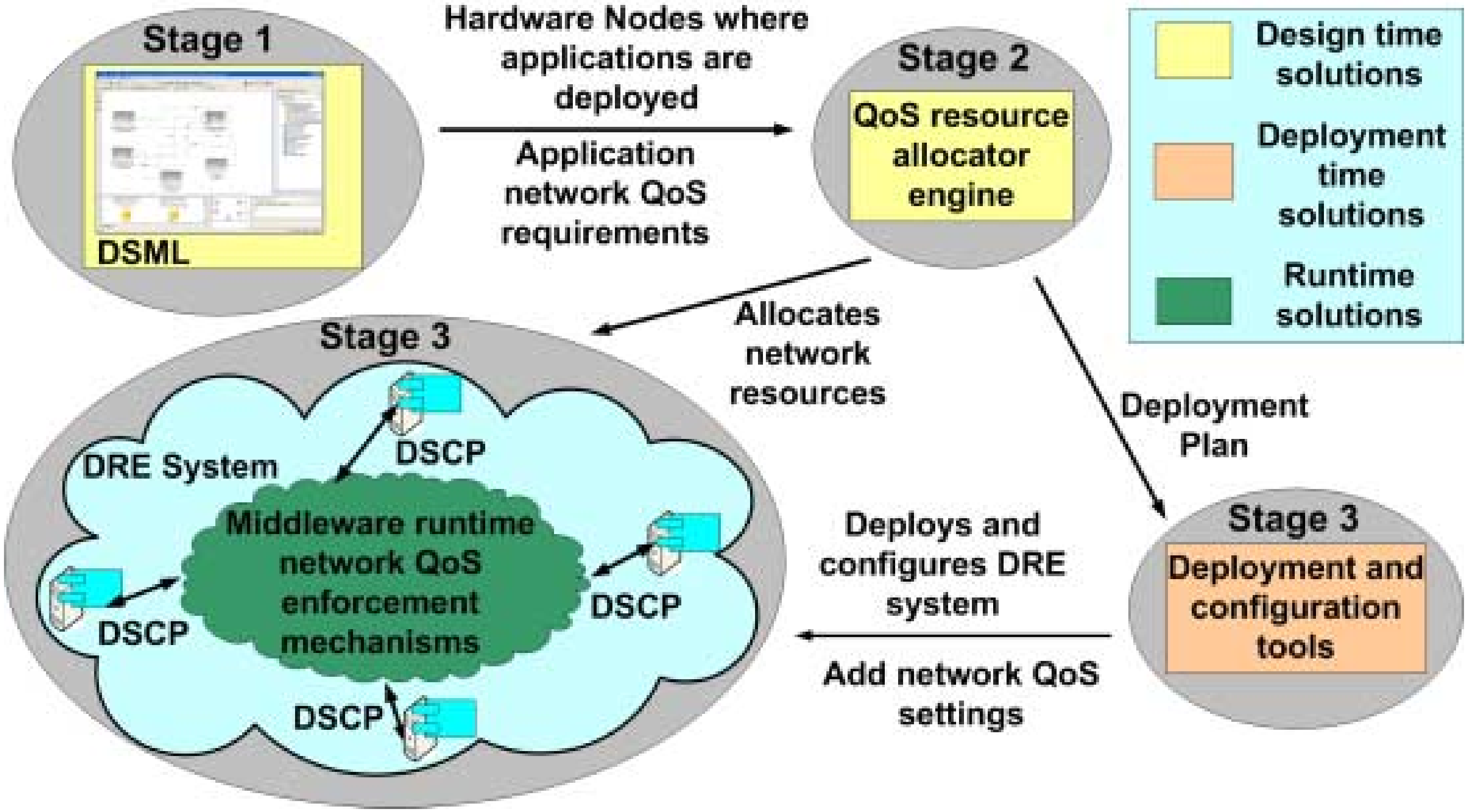
- Allocate and configure network resources and devices
  - availability of resources
  - per-hop-behavior configurations at routers
- Extensible network resource allocation
  - Requirements vary depending on the deployed contexts
  - Mechanisms vary depending on deployed contexts
- Problem
  - application code modification to work with network QoS mechanisms
    - No awareness of usability context
    - tied down to a particular network QoS mechanism
    - On-demand use of network QoS mechanisms

Challenge 3: Runtime Network QoS Settings



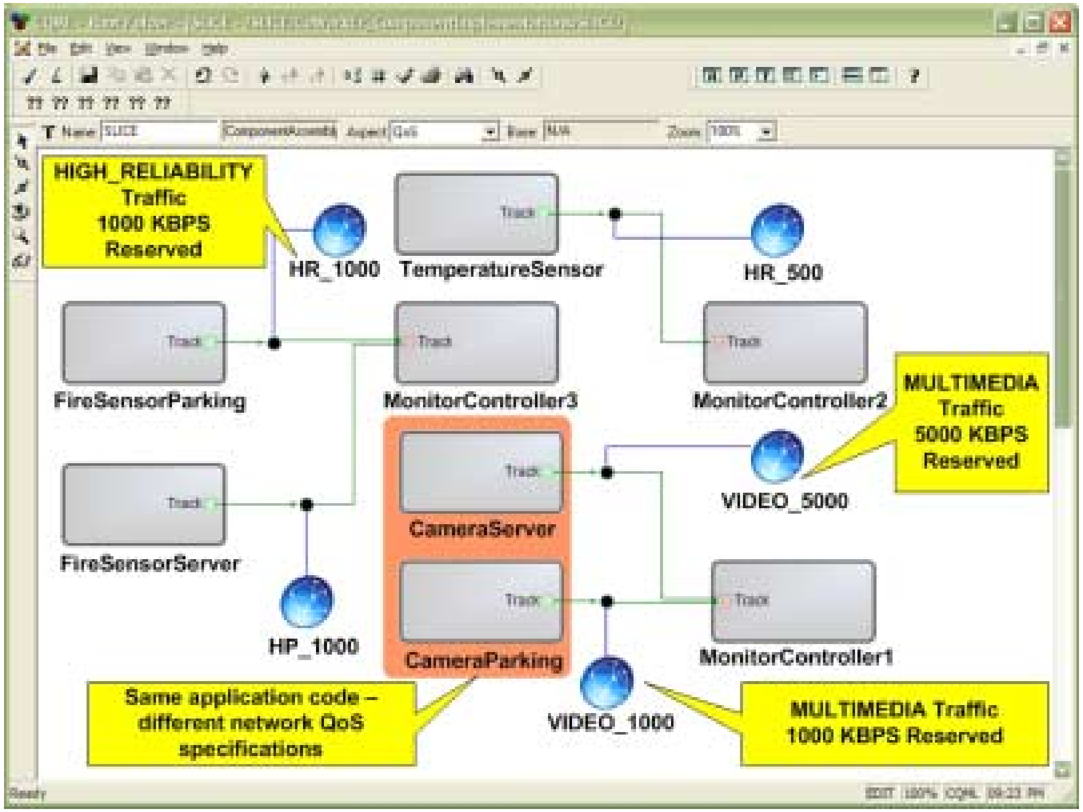
- Application remote communications
  - using chosen network QoS settings (e.g., DSCP markings)
  - network layer differentiation
  - written code to instruct the middleware to add network QoS settings on behalf of the applications
- DRE application development process
  - no awareness to the deployed context
  - focus on business logic rather than write code to provision QoS
- Problem
  - manual or programmatic specification of network QoS settings
  - tedious
  - error-prone
  - non-scalable

NetQoPE Multistage Architecture



- NetQoPE provides a three-stage solution for the three challenges described
- Stage 1
  - Capabilities for intuitive and scalable network QoS specification
- Stage 2:
  - Capabilities for resource allocation and configuration
- Stage 3:
  - Capabilities for runtime support for QoS settings enforcement

Stage 1 : Model Driven Engineering

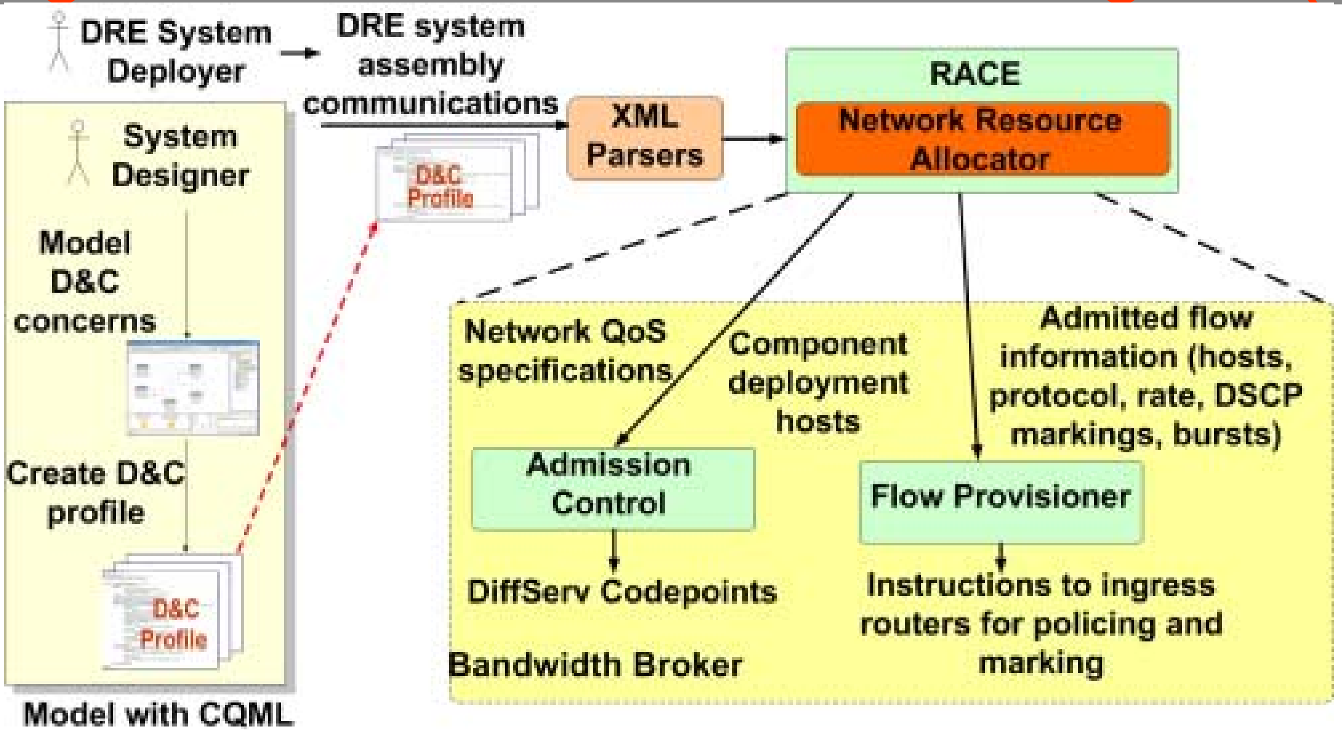


Office Scenario

- same application – different network QoS class and requirements
- same network QoS class – different requirements
- multiple network QoS class specifications

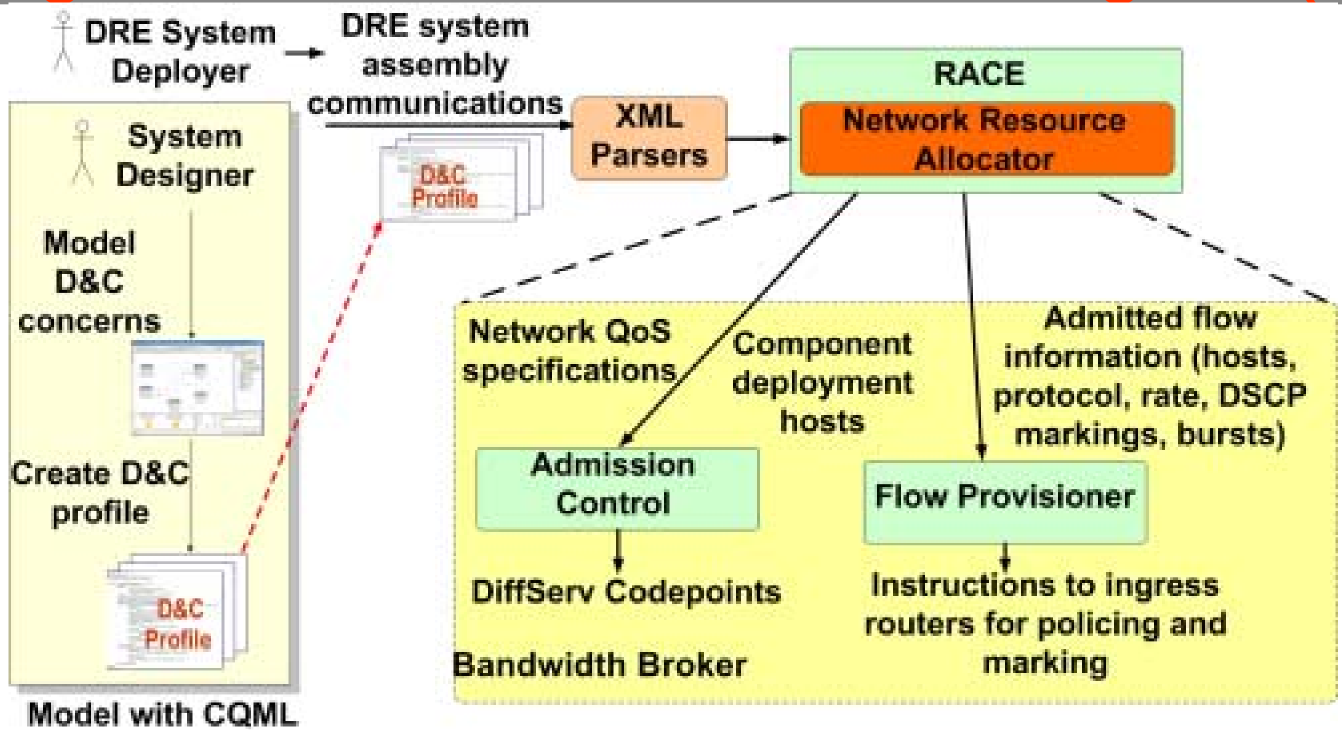
- Model Driven Engineering solution
  - Component QoS Modeling Language (CQML)
  - Provides intuitive abstractions to specify QoS
  - Scalable solutions
  - Developed in GME
- Network QoS modeling allows modeling QoS per application flow
  - Classification into high priority (HP), high reliability (HR), multimedia (MM) and best effort (BE) classes
  - Enables bandwidth reservation in both directions
  - Client propagated or server declared models

Stage 2: Resource Allocator Engine (1/3)



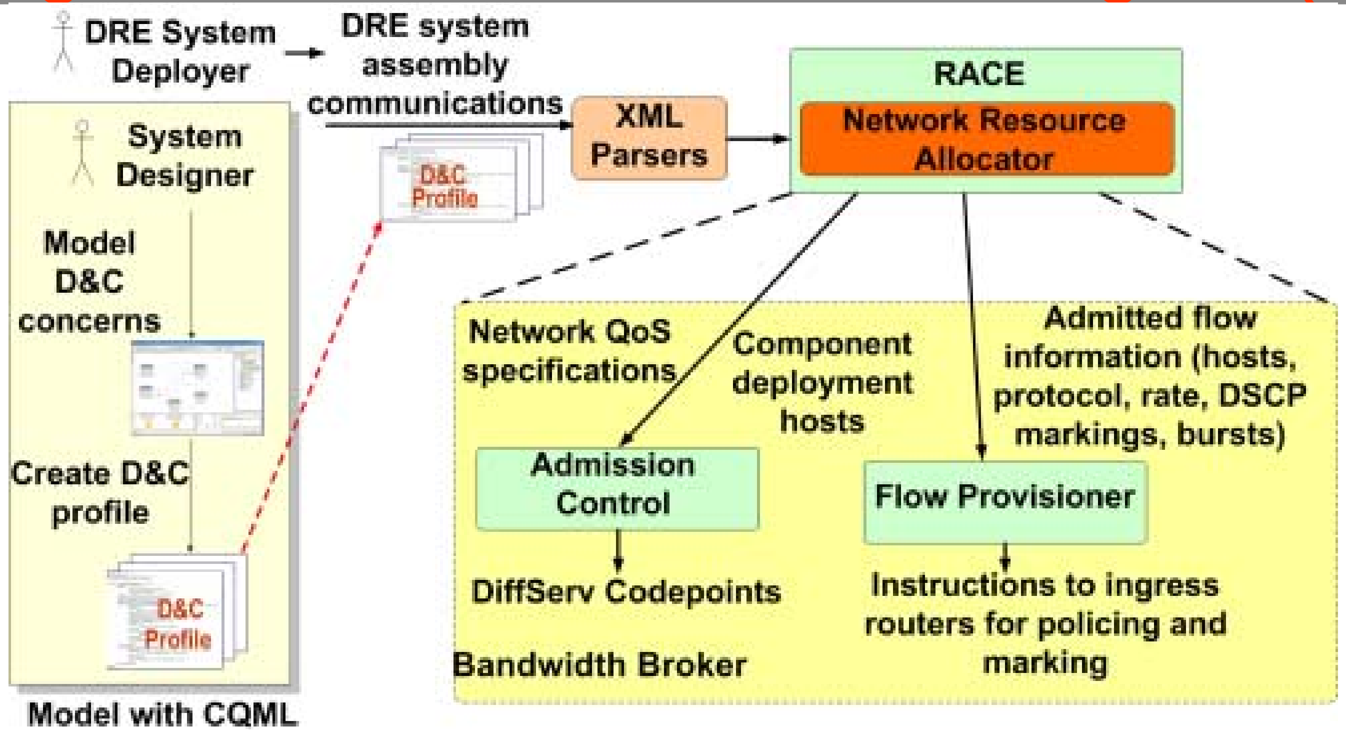
- Resource allocation framework – RACE network resource allocator
  - pluggable network resource allocators
  - specific solution – DiffServ Bandwidth Broker from Telcordia Technologies
- Deployment descriptors from CQML
  - end-to-end application flows requiring QoS requirements
  - allocation done only when necessary
  - context-specific allocations

Stage 2: Resource Allocator Engine (2/3)



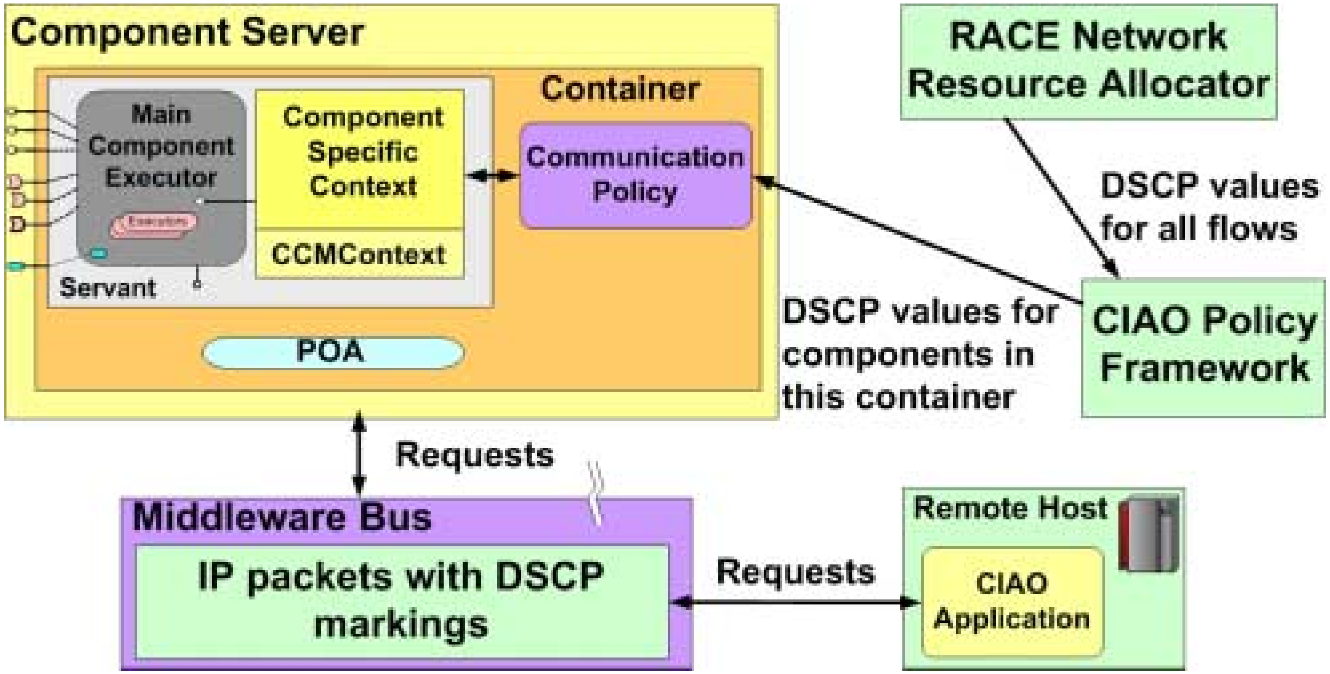
- Two-phase resource allocation and configuration
- Admission control capabilities
  - allocation of resources one-flow at a time
  - preferences given for flows with more importance
  - provide opportunity to change implementations or change deployment
    - resources not available
    - degraded QoS planning at design time rather than at runtime

Stage 2: Resource Allocator Engine (3/3)



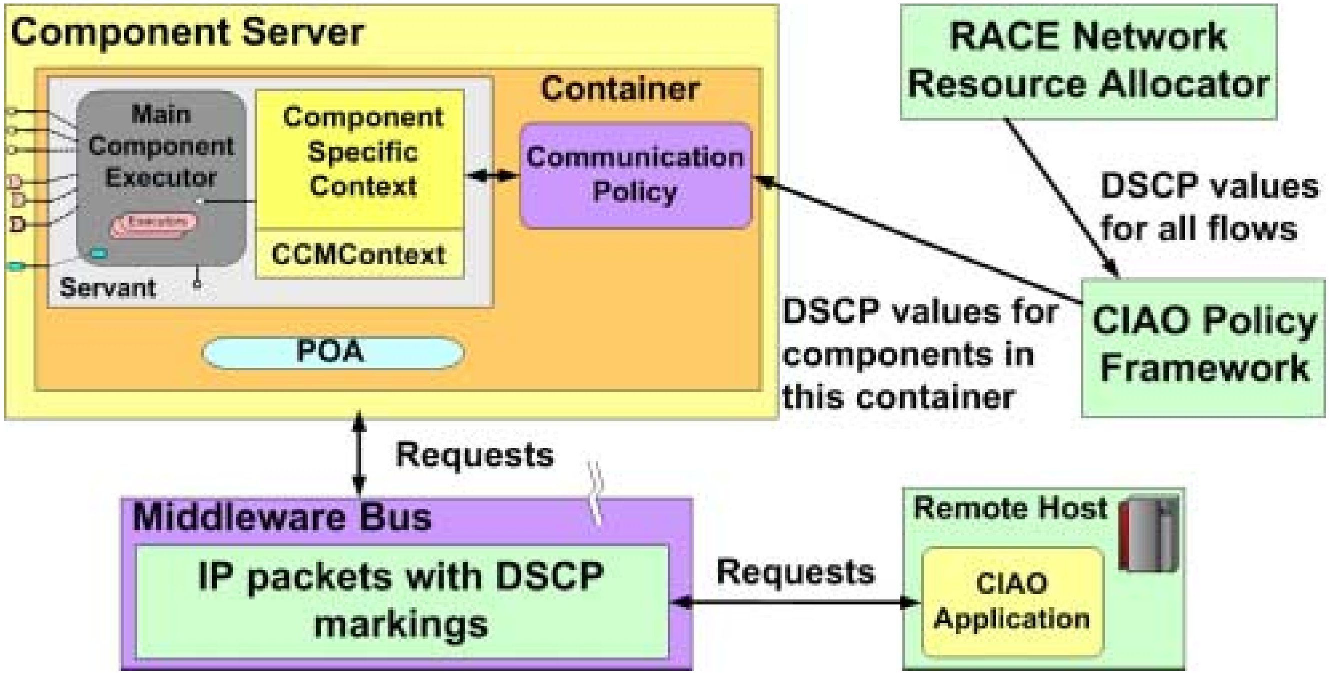
- Network device configuration
  - Flow provisioner – hides network device API differences and complexities
  - per-hop behavior at routers
  - allocation of DiffServ codepoints to be used by application remote communications
  - deployment descriptors used for deploying applications with needed network QoS

Stage 3: Runtime Policy Framework (1/3)



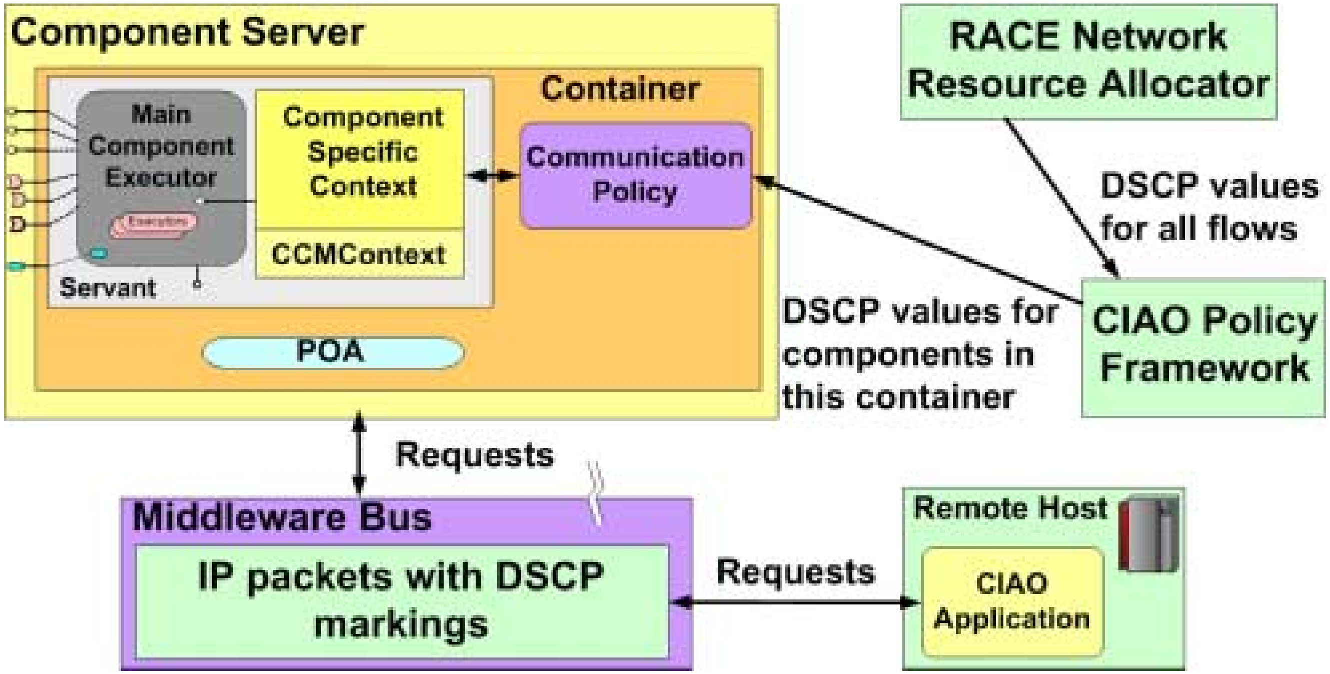
- Network settings configurator -- DAnCE
  - auto-configure applications with the network QoS settings specified in the deployment descriptors
  - specification of middleware policies to configure underlying middleware to add network QoS settings
    - e.g., forward and reverse (request/reply) DSCP markings to IP packets

Stage 3: Runtime Policy Framework (2/3)



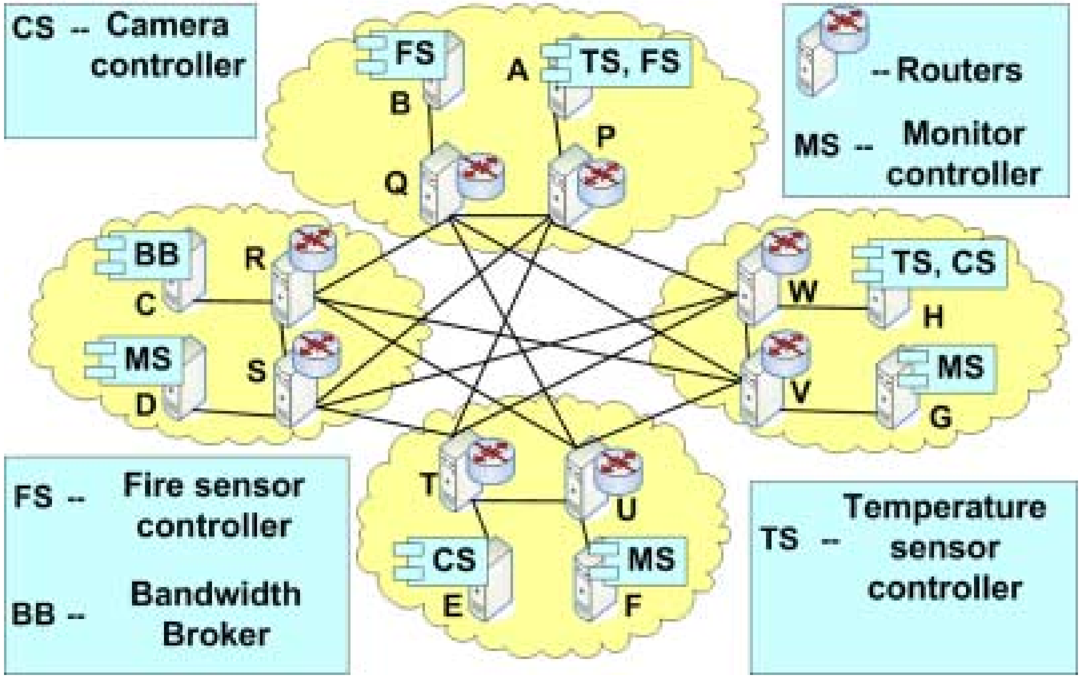
- Enforcement of network QoS settings – CIAO middleware
  - appropriate network QoS settings provisioned to applications using object references
  - application transparent
    - no extra code written
    - object references encoded with policies to add DSCP markings
    - container programming model

Stage 3: Runtime Policy Framework (3/3)



- Client propagated network policy model
  - clients dictate the forward and reverse DSCP markings to be used
  - reverse DSCP marking to be used added as a service context in the request
- Server declared network policy model
  - servers dictate both the forward and reverse DSCP markings to be used
  - handshake between client and server

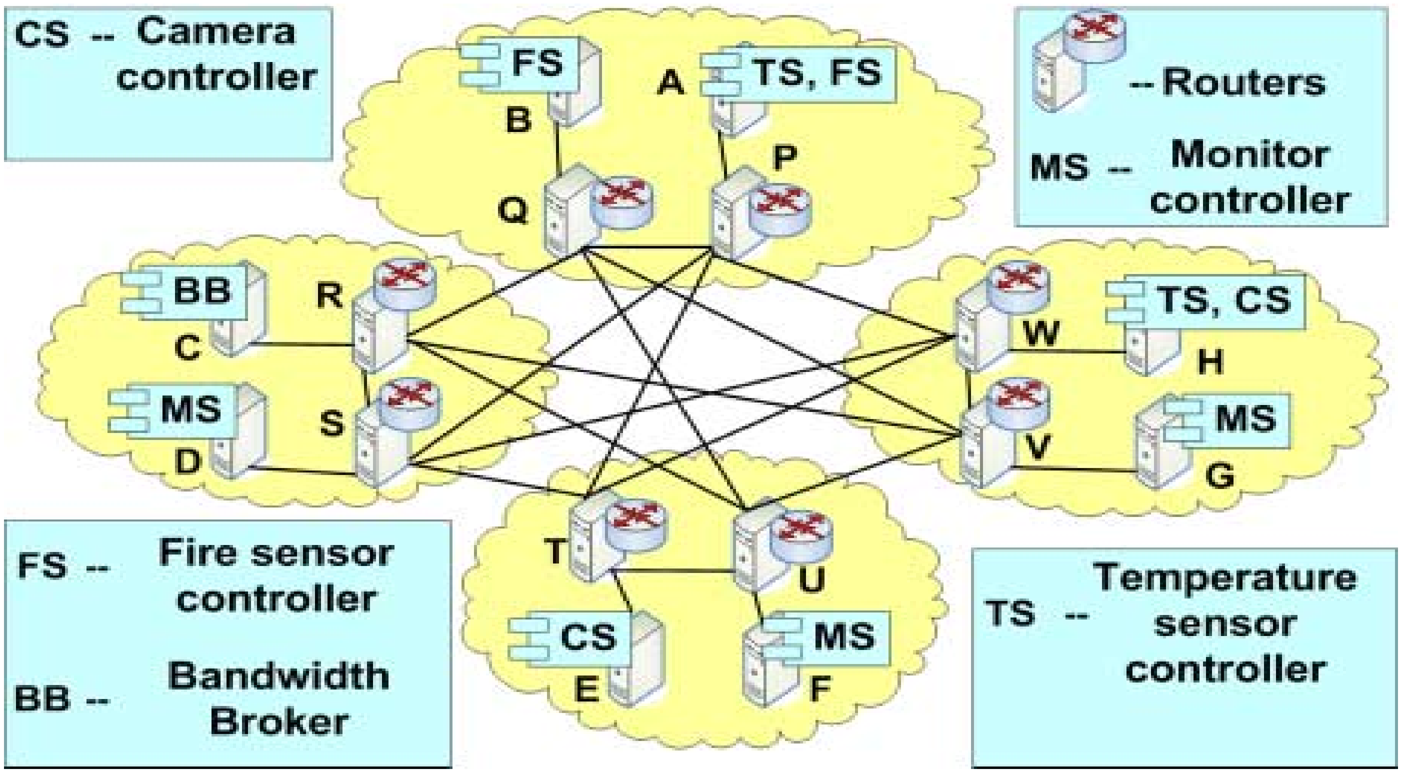
Evaluating NetQoPE



- Experimental Setup
  - ISISlab setup blade servers running Fedora core
  - DiffServ QoS over IP Networks
  - Telcordia Bandwidth Broker
  - CIAO Applications deployed over different blades
  - Linux router software

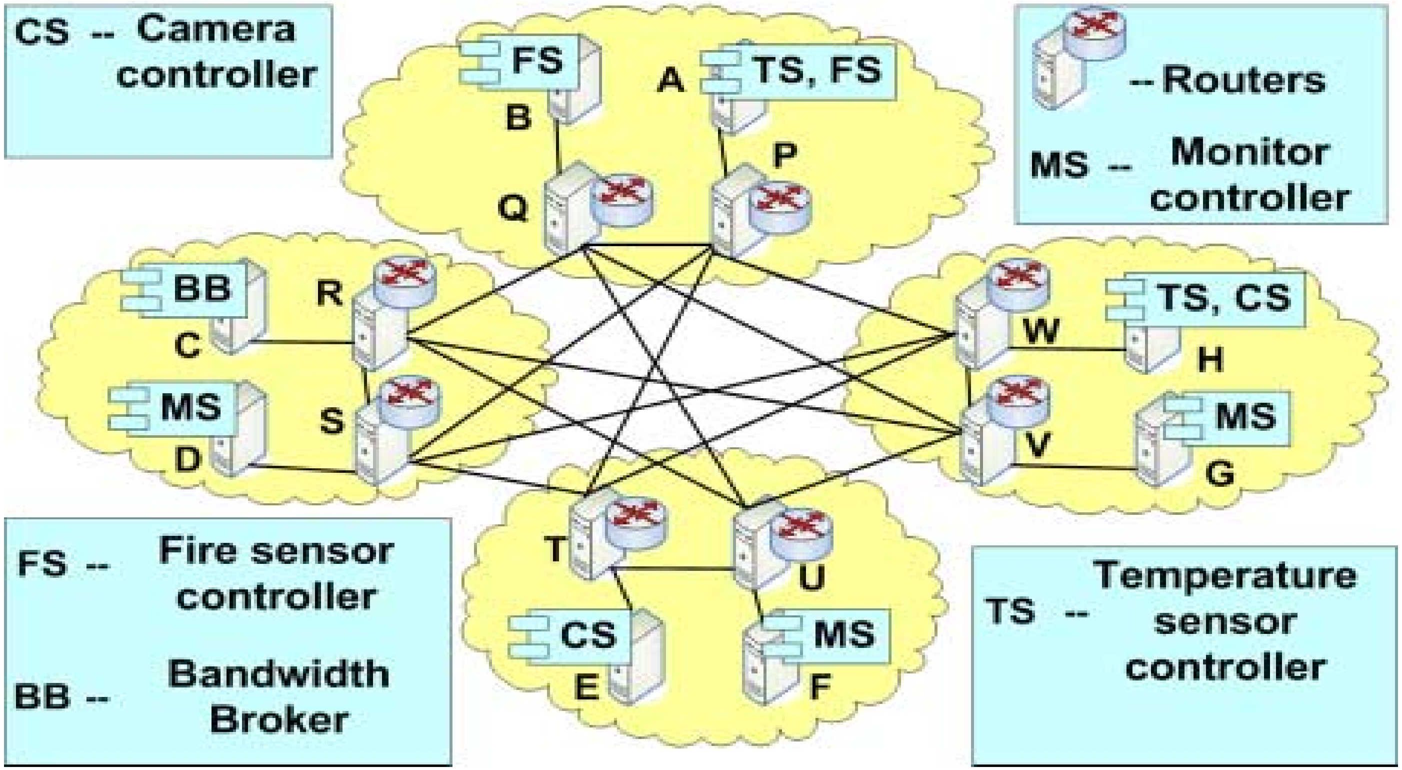
- Evaluation objectives
  - Overhead of the runtime middleware adding network QoS settings when applications make remote communications
  - QoS customization capabilities for different applications
  - admission control capabilities

Results 1: Measuring Runtime Overhead (1/3)



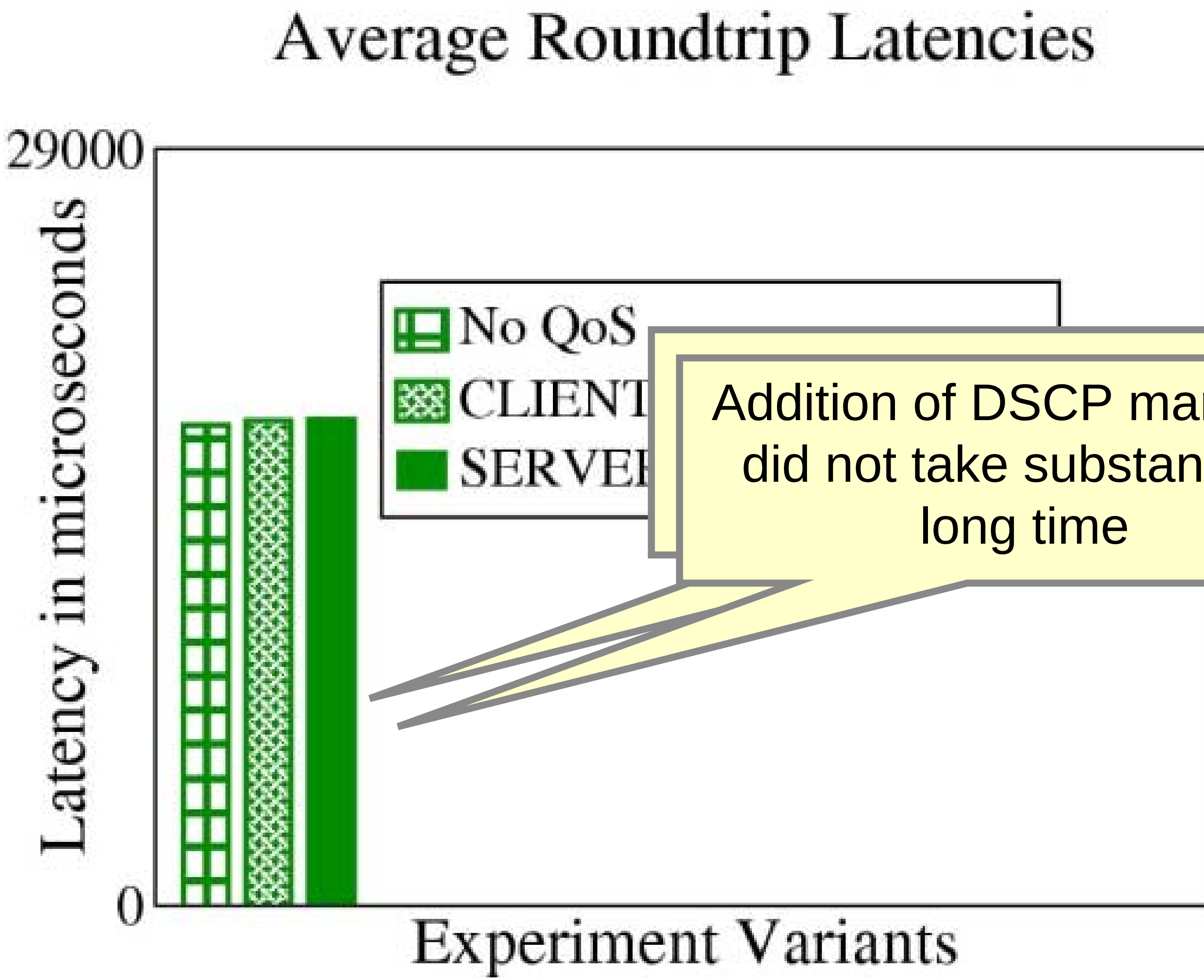
- Rationale – NetQoPE's Overhead measurements
  - CQML, RACE network resource allocator, and network configurator used before runtime
  - no runtime overhead
  - runtime overhead added by the CIAO middleware to add DSCP markings in both the network policy models

Results 1: Measuring Runtime Overhead (2/3)



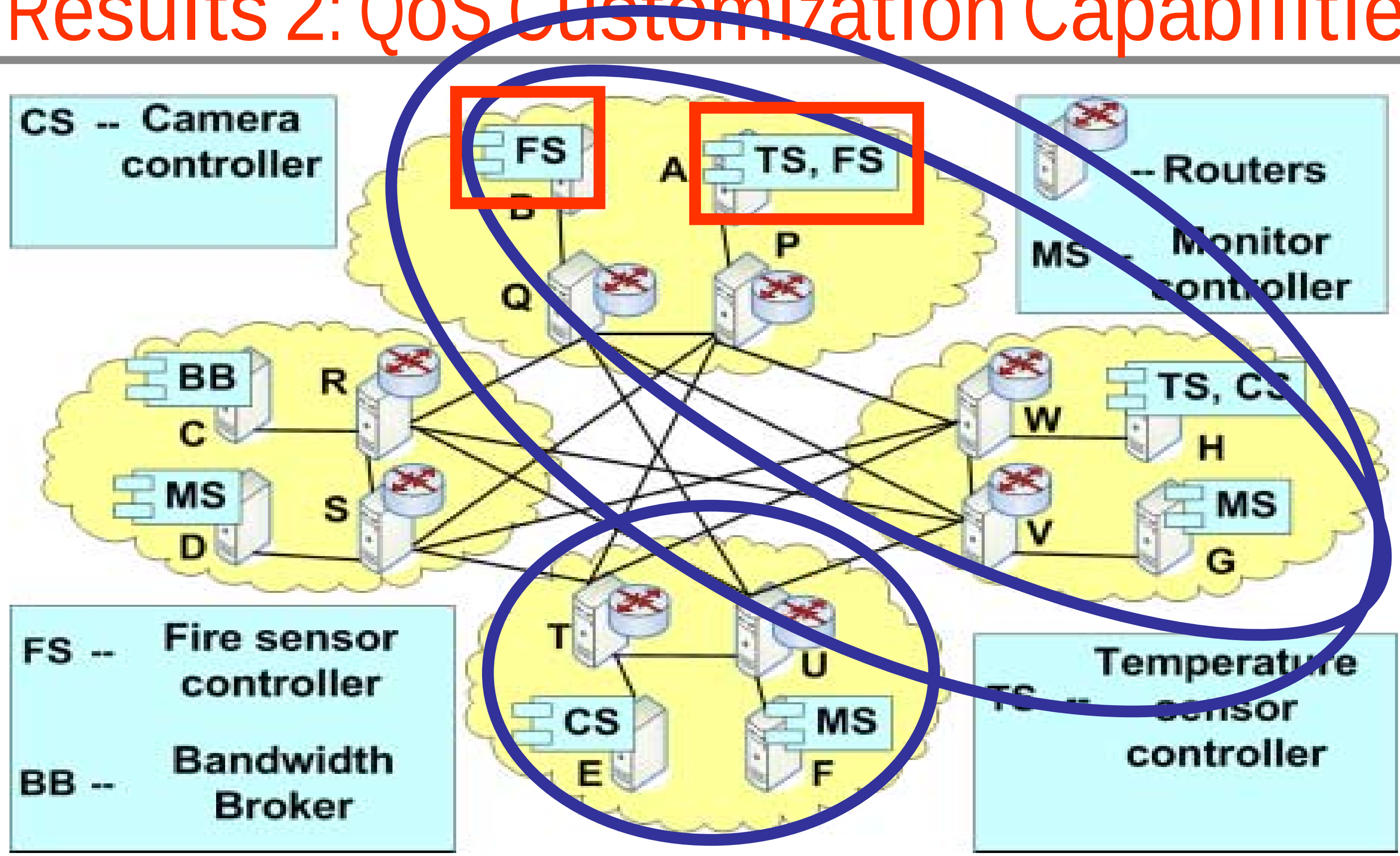
- Experiment methodology – measure average invocation latency
  - client and server with no network QoS
  - client and server with network QoS
    - client propagated network policy model
    - server declared network policy model

Results 1: Measuring Runtime Overhead (3/3)



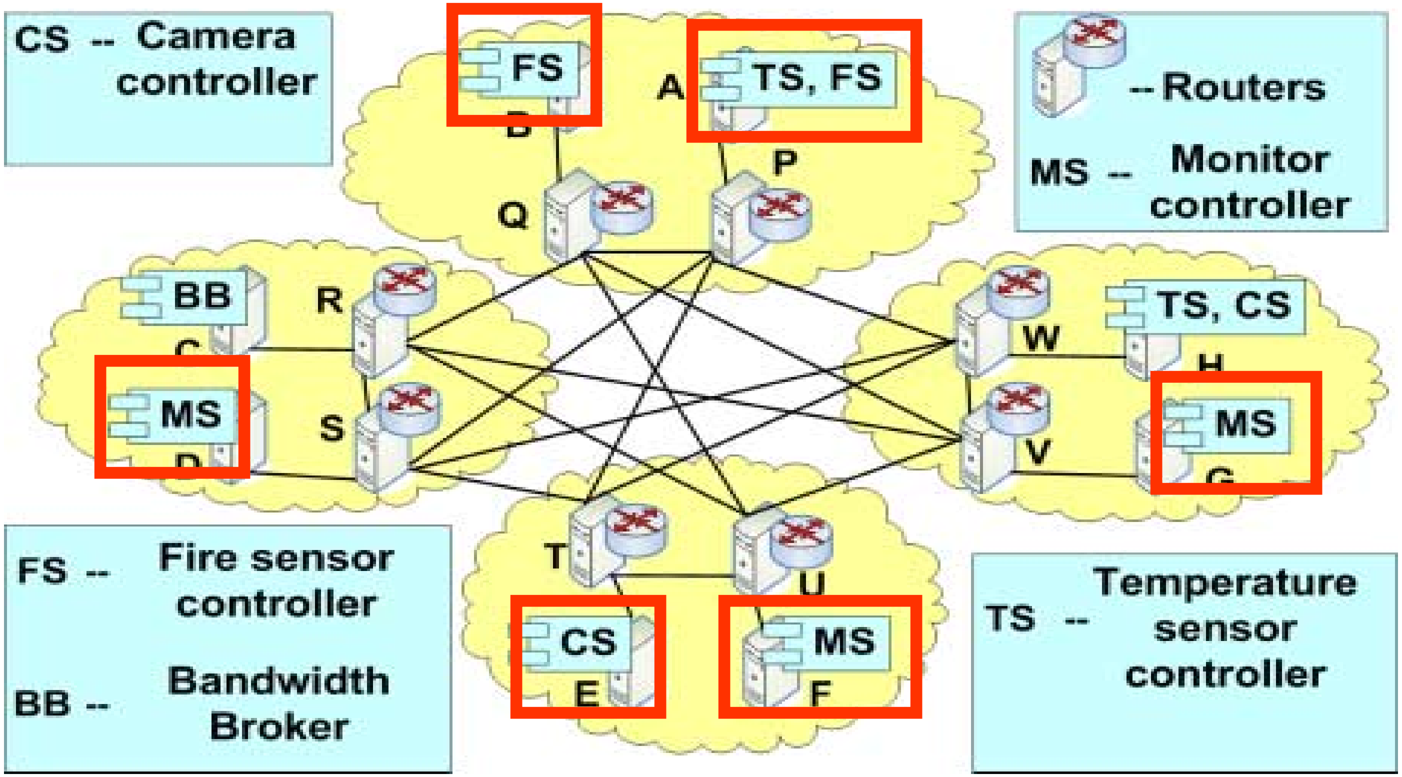
Addition of DSCP markings did not take substantially long time

Results 2: QoS Customization Capabilities (1/4)



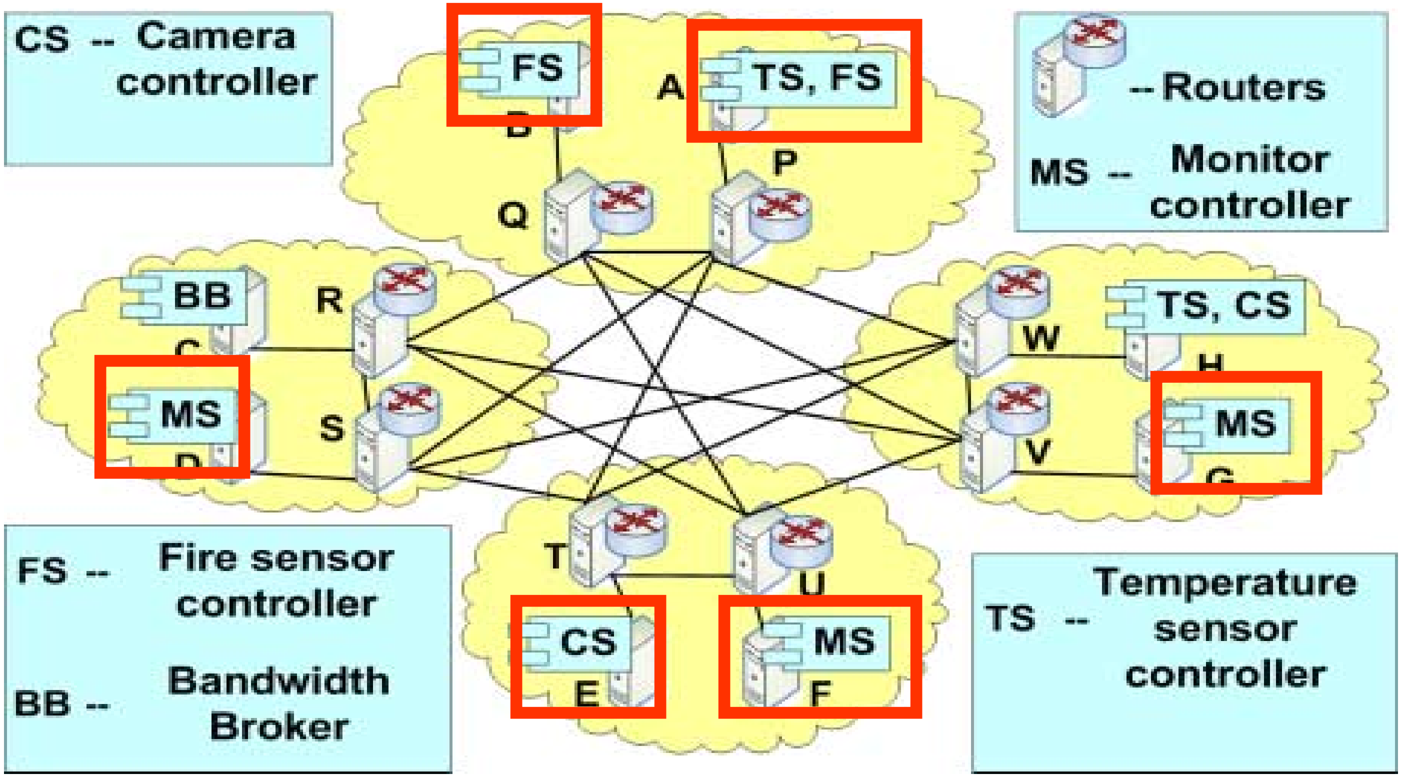
- Rationale – evaluate NetQoPE's QoS customization capabilities
  - provisioning different network QoS requirements
  - same application code – different QoS requirements
  - same network QoS class – different bandwidth requirements

Results 2: QoS Customization Capabilities (2/4)



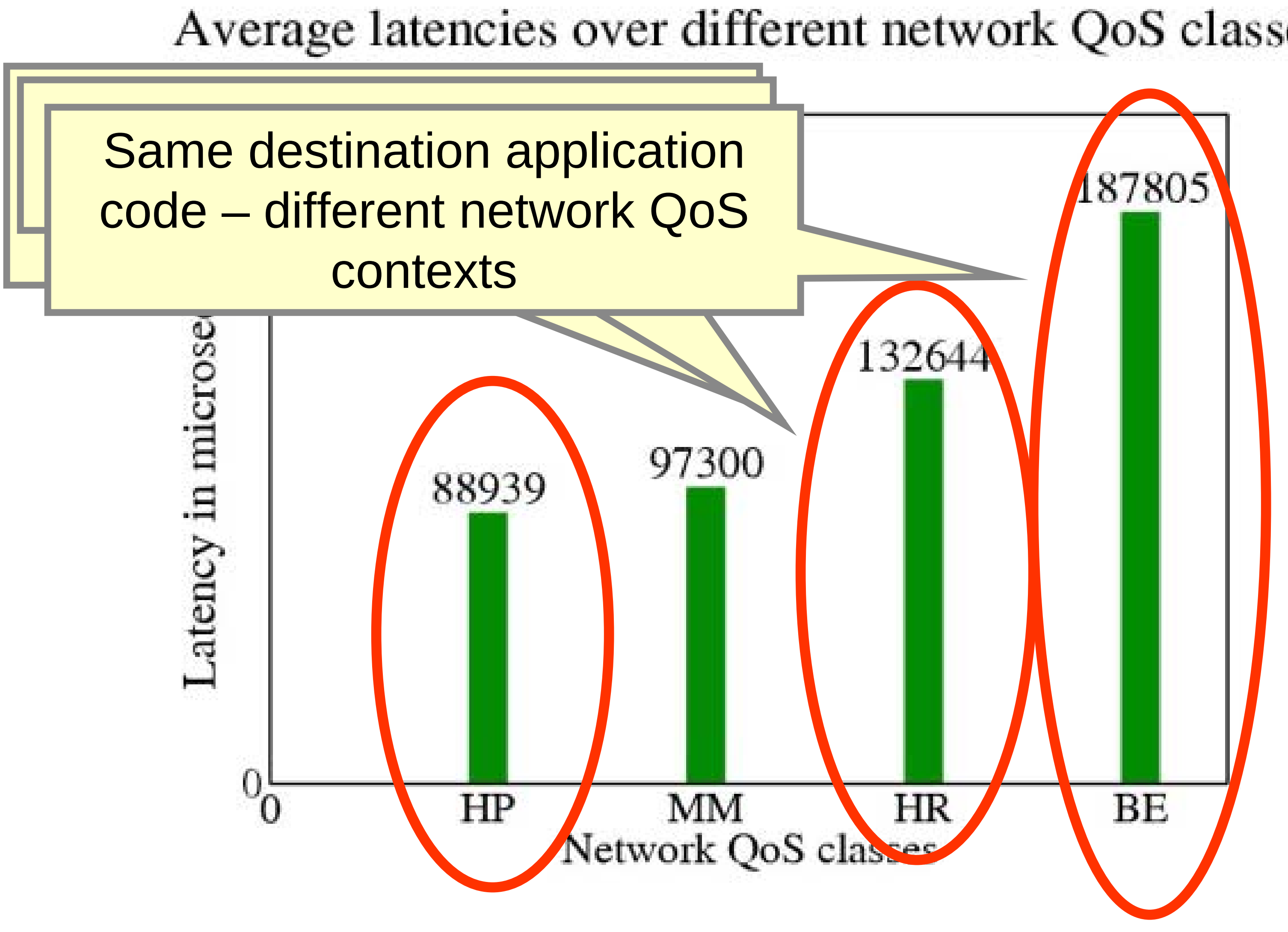
- Methodology – measure average invocation latency
  - high priority network QoS class
  - high reliability network QoS class
  - multimedia network QoS class
  - best effort network QoS class
  - different QoS requirement contexts

Results 2: QoS Customization Capabilities (3/4)

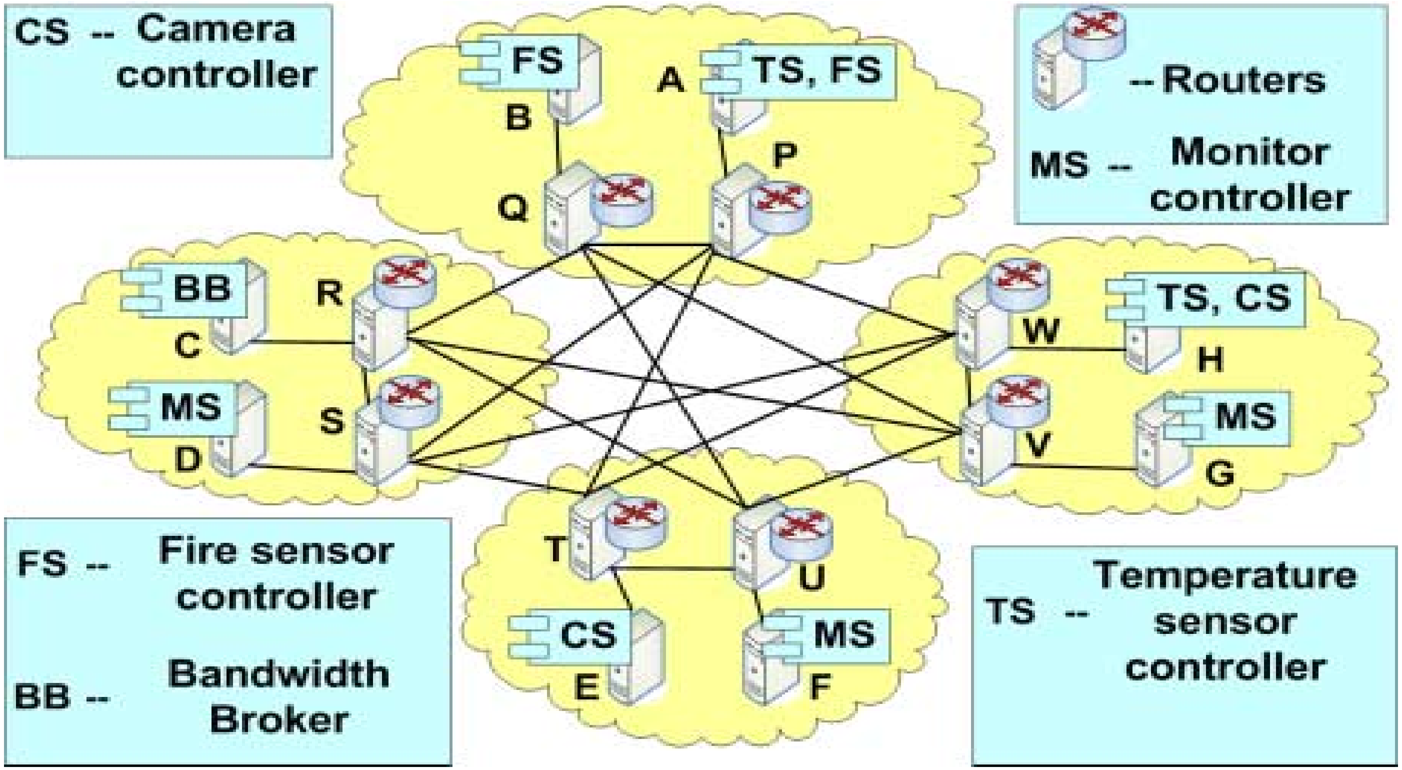


- Methodology
  - all links 100 Mbps
  - background network loads
  - sufficient link capacity in each of the links for each of the network QoS classes
  - resources allocated and application communications with DSCP markings

Results 2: QoS Customization Capabilities (4/4)

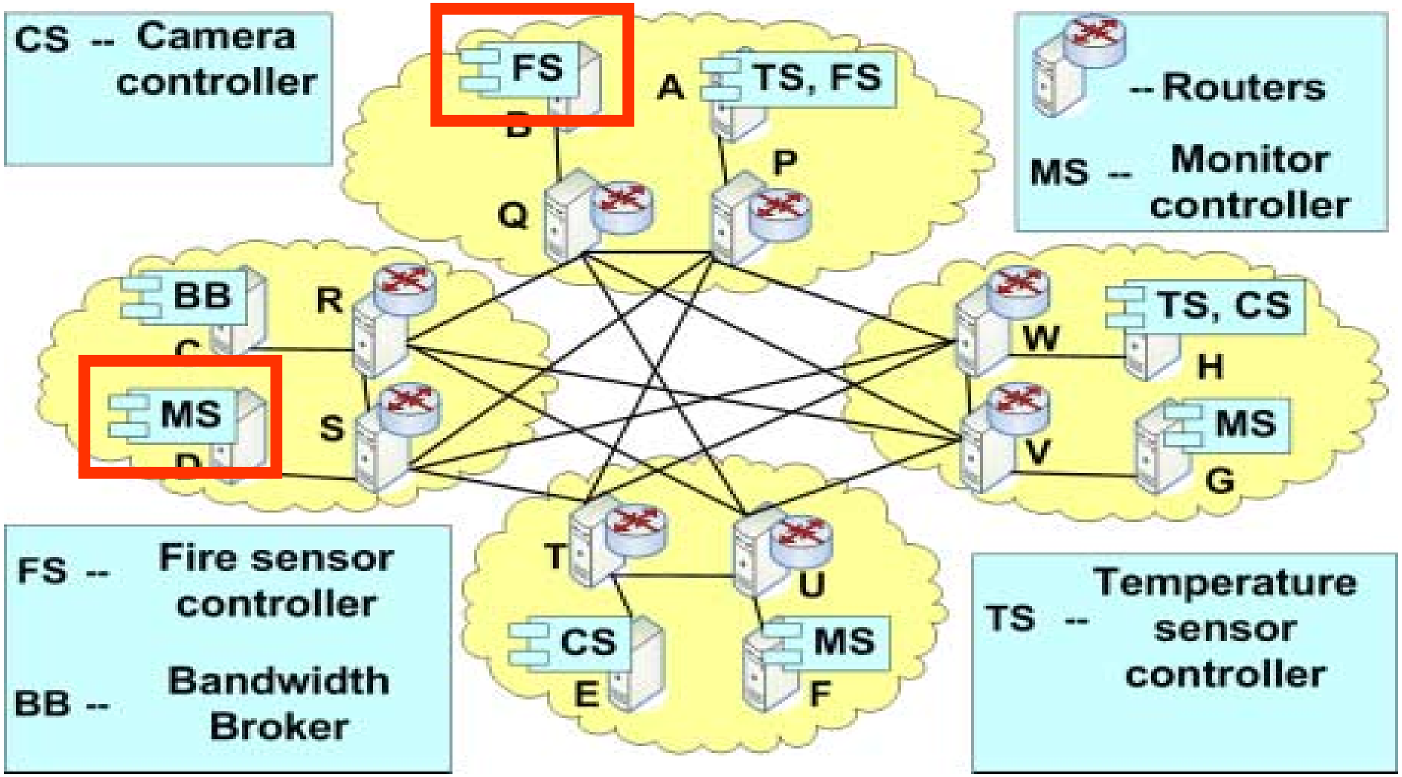


Results 3: Resource Allocation Capabilities (1/5)



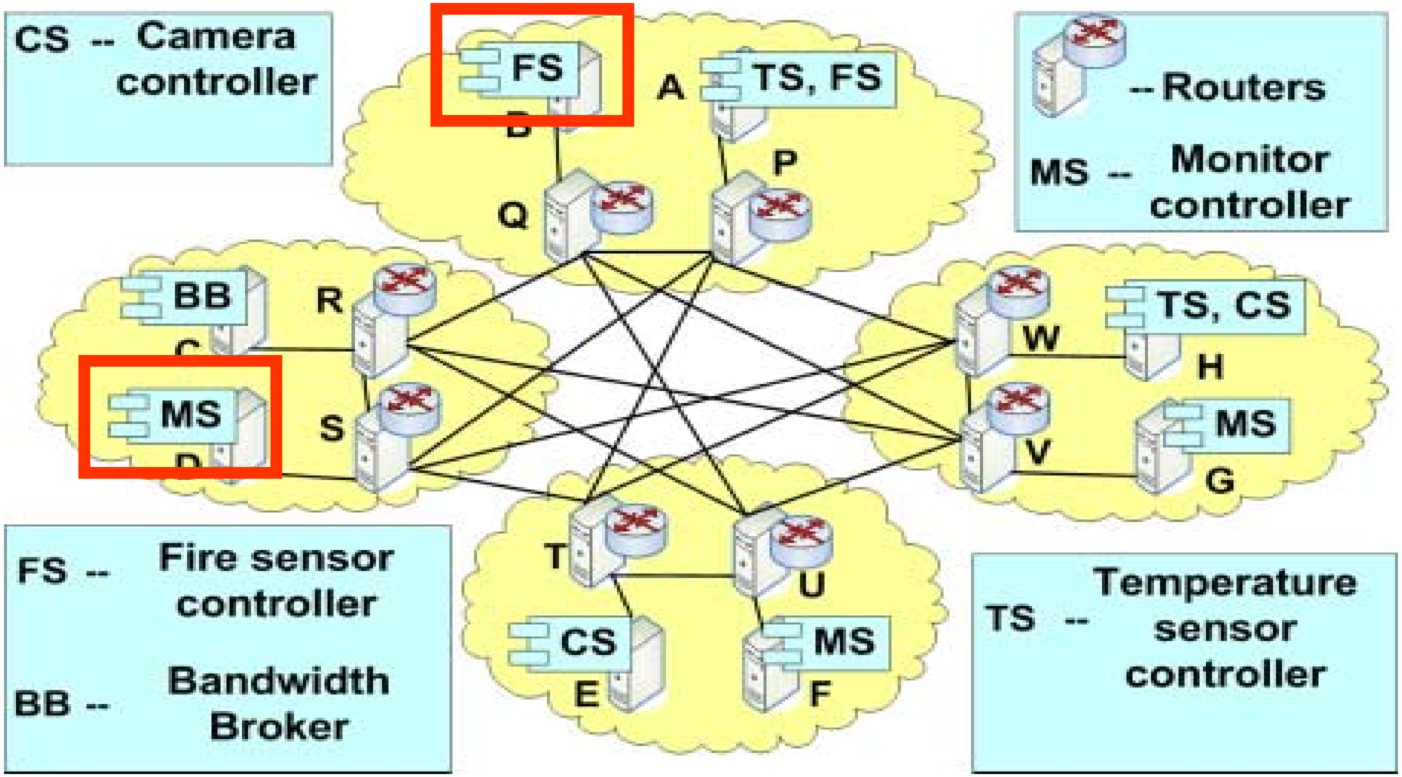
- Rationale – evaluate NetQoPE's resource allocation capabilities
  - two-phase resource allocation process
  - non-availability of network resources provides opportunities to change deployment contexts
  - degraded QoS decisions before applications are running

Results 3: Resource Allocation Capabilities (2/5)



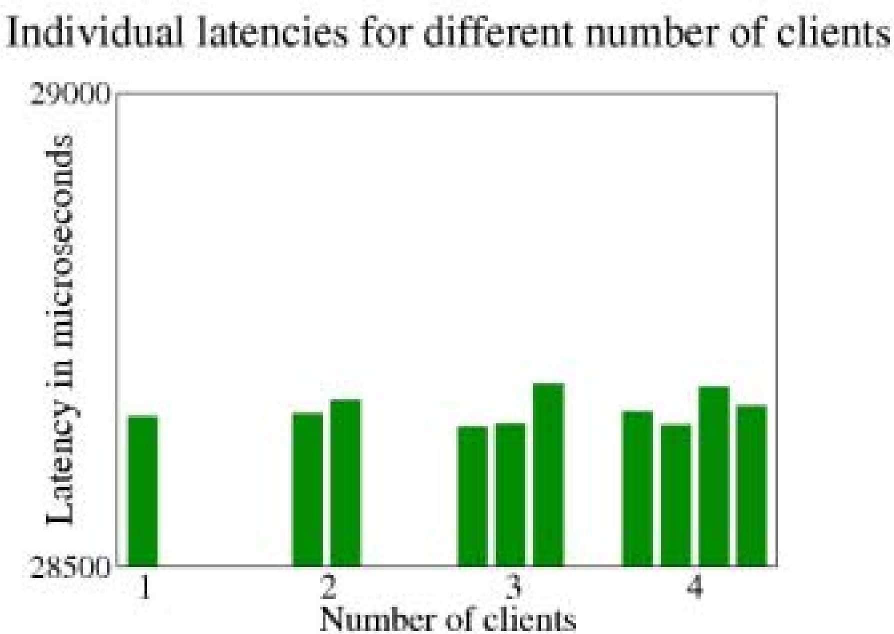
- Methodology – how many pairs can be deployed?
  - multiple pairs of application components between hosts B and D using high reliability (HR) network QoS class
  - each pair requests 6 Mbps forward and reverse bandwidth
  - total link capacity for HR class – 30 Mbps
  - background traffic in other classes saturated

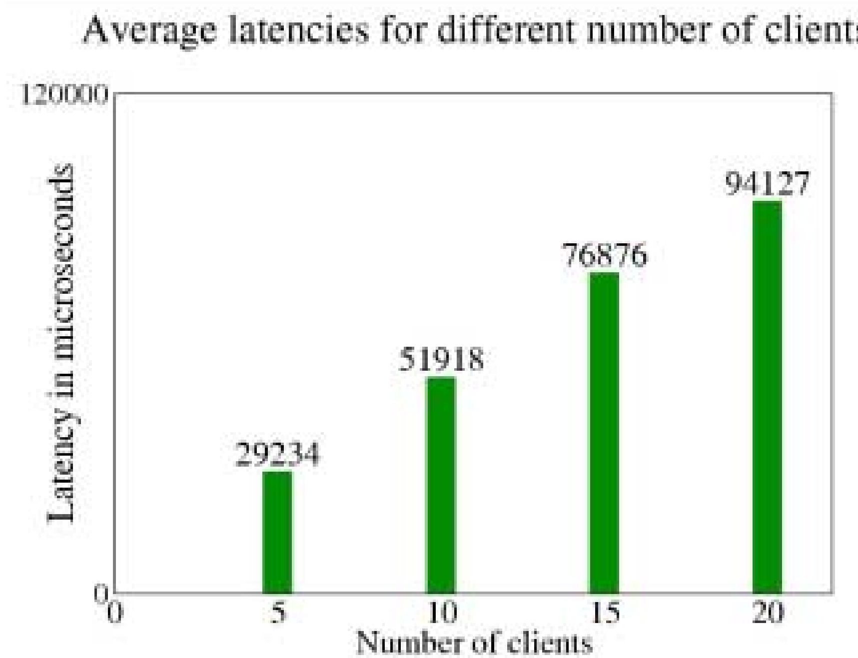
Results 3: Resource Allocation Capabilities (3/5)



- Methodology – how many pairs can be deployed?
  - deploying up to 4 pairs, admission control capabilities were used
  - more than 4 pairs, admission control capabilities were not used
    - first phase of the two phase not used
    - just router configuration was done, and DSCP markings to be used provided

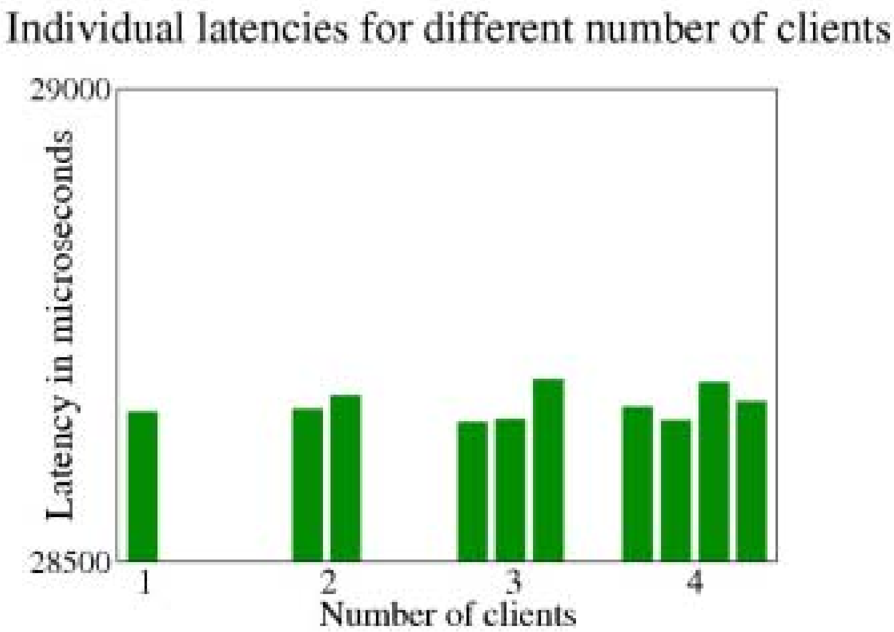
Results 3: Resource Allocation Capabilities (4/5)

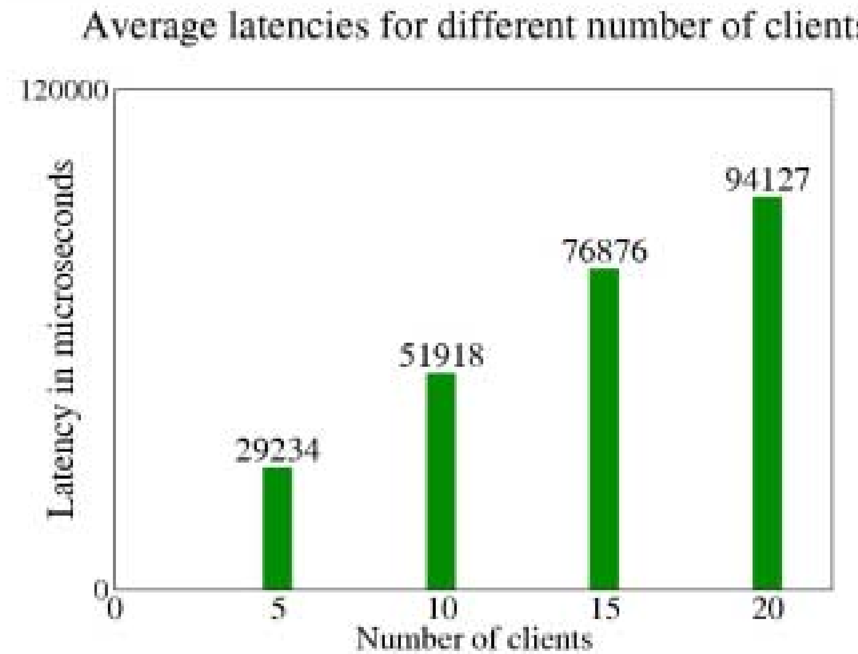




- Average invocation latency
  - average latencies similar for all the application communications when admission control is used
  - average latencies vary a lot when admission control was not used.
  - application logic totally decoupled from working with network QoS mechanisms

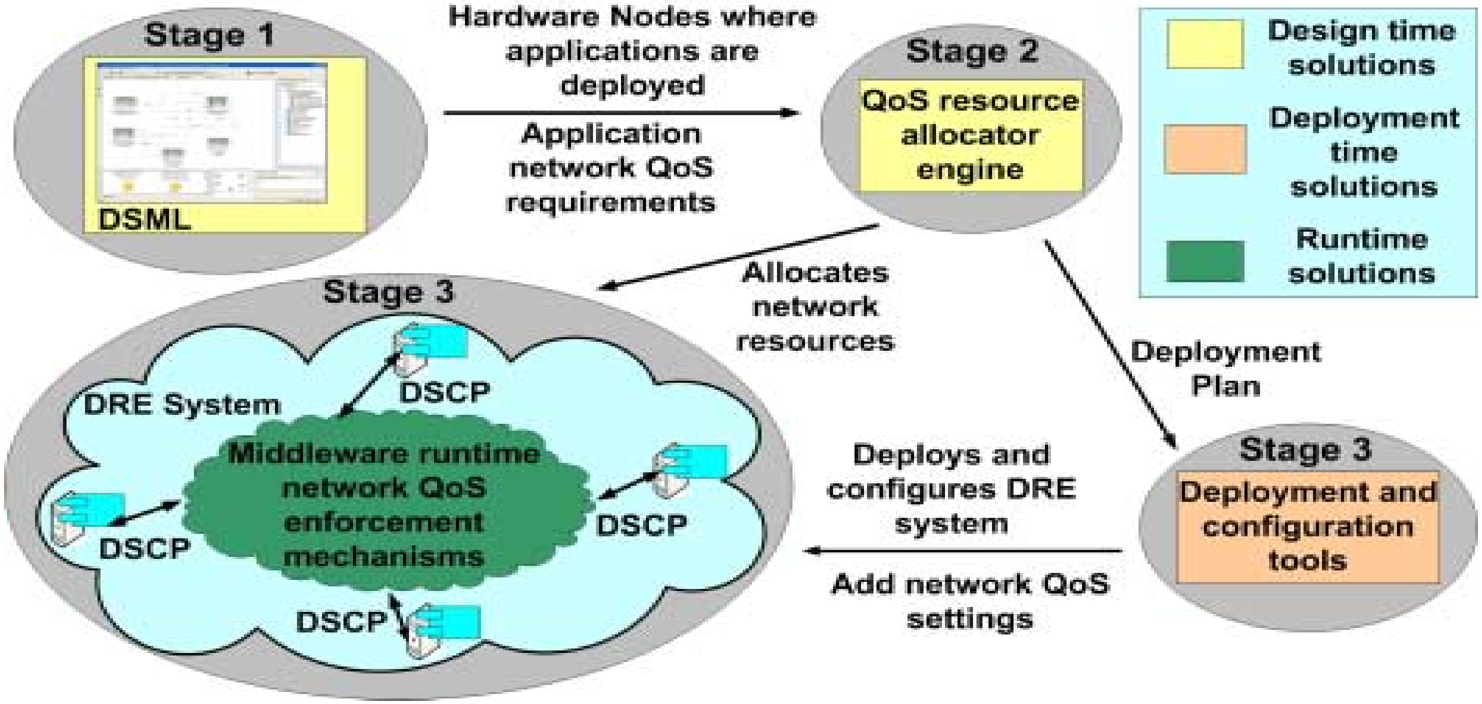
Results 3: Resource Allocation Capabilities (5/5)





- No admission control
  - even deserving applications suffer
  - require runtime application adaptations
  - deployment/design time admission control would have prevented more than 5 applications
    - degraded QoS in the case of more than 5 applications

Concluding Remarks



- Different stages required to address the variabilities in the problem and solution space
  - NetQoPE provides a combination of design/deployment/runtime solutions to provide network QoS provisioning for applications
  - application-transparent solutions providing opportunities to support many QoS deployment contexts

NetQoPE can be downloaded from : www.dre.vanderbilt.edu