Next Generation Middleware Support for Mobility

David L. Levine
Washington University, St. Louis
levine@cs.wustl.edu

8 March 2000

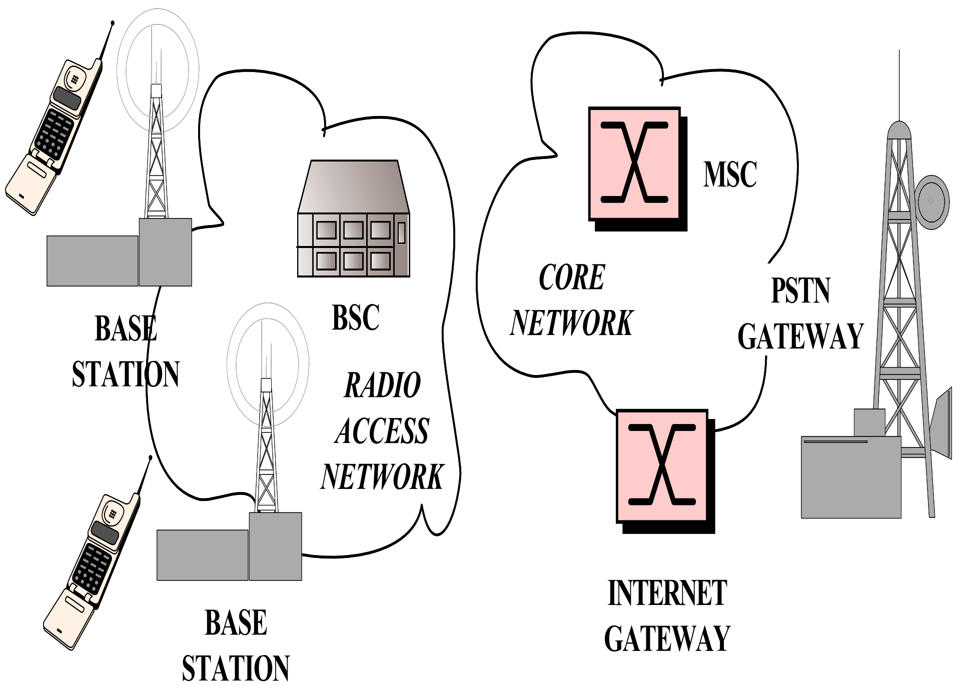
http://www.cs.wustl.edu/~levine/research/srs00.ps.gz

Sponsors: Sprint, Siemens, Nortel, NSF, Motorola, Lockheed Martin,
Hughes Network Systems, DARPA, and Boeing

Overview

- Motivation and context
- Middleware to support mobility
  - Features
  - State-of-the-art
- Next generation mobility middleware features
- Mobility middleware design guidelines

Pervasive Web Access

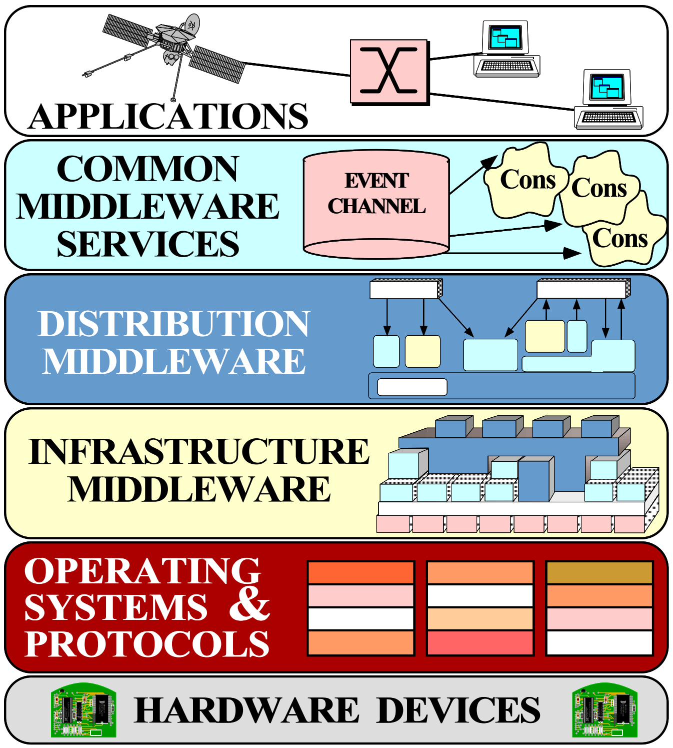


- Web use is often, but not always, anonymous
- Asymmetric throughput requirements, *i.e.*, mostly download
- Dependability is important: 0 downtime
- Latency and predictability (jitter) requirements not usually stringent

Motivation for Middleware

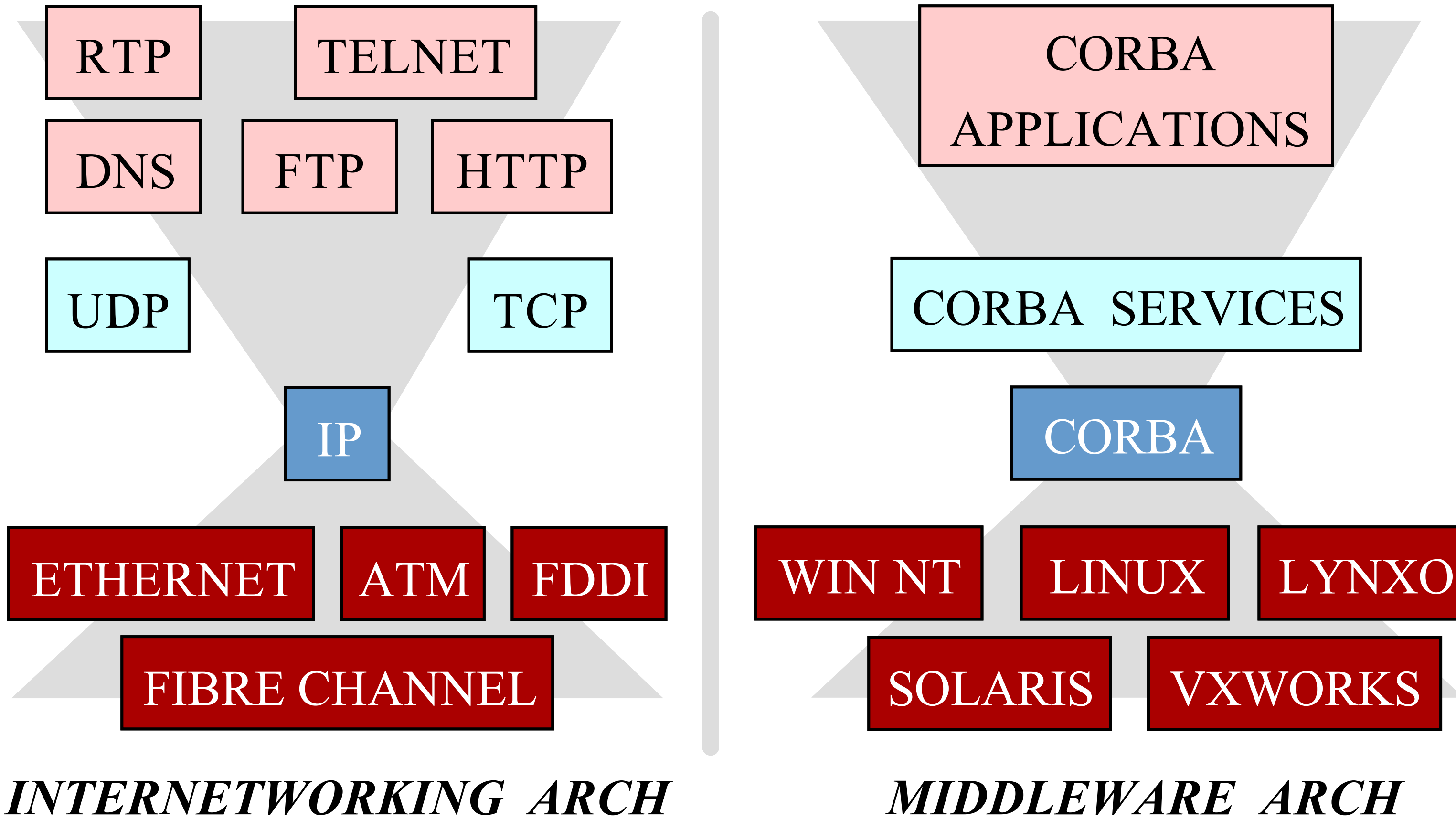
- Reduced time-to-market and development cost
  - Unpredicted/unanticipated web usage indicates short client product lifetimes and requires flexible/adaptable servers
- Middleware eases object-oriented software development
  - Provides location transparency
  - Provides language/platform independence
  - Provides modularity
  - Provides robustness

Context: Levels of Abstraction in Software



- **Observations**
  - Historically, distributed apps built directly atop OS
  - Today, more and more apps built atop *middleware*
  - Middleware has several layers
- **Decision Points**
  - Buy vs. build
  - Identify reuse boundaries
  - Determine where to add value

Context: Levels of Abstraction in Internetworking and Middleware



Mobility Issues

- Physical mobility
  - Host movement, relative to other hosts (and network)
  - Defines (dynamic) target execution environment
- Logical mobility
  - Code and data movement between hosts
  - Permits dynamic application/host component bindings
- Coordination
  - Includes mechanisms for peer discovery, information exchange, and cross-host synchronization

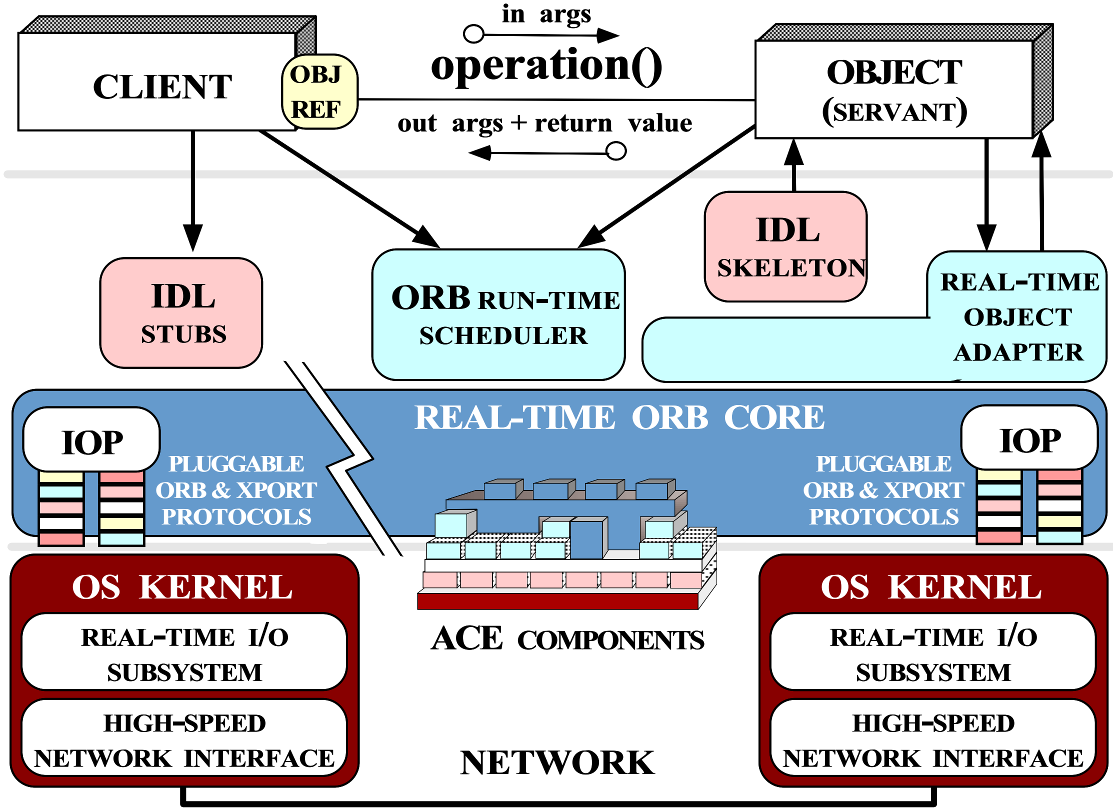
Mobility Middleware Requirements

- For physical mobility
  - Specify context for an application
  - Detect location changes
  - Associate location changes with context changes
  - Determine application effects of context changes
- For logical mobility
  - Code and data migration support
  - Clean model with robustness and security

Can Middleware Perform?

- It must offer low overhead . . .
  - middleware/endsystem CPU overhead must be low
  - (OS context switch time must be low)
- Priority inversion must be eliminated . . .
  - to provide QoS to high priority requests
- Predictability must not impair application performance
- Middleware overhead must be low
- Memory footprint must be minimized

Implementation: The ACE ORB (TAO)

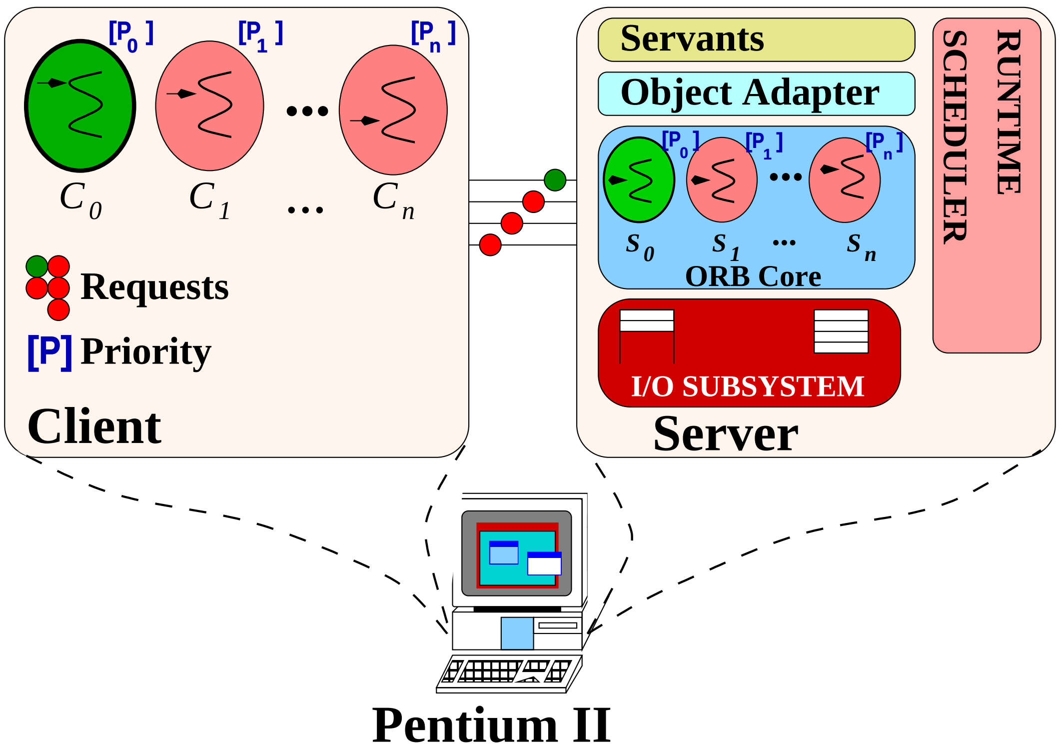


`http://www.cs.wustl.edu/~schmidt/TAO.html`

TAO Overview →

- An open-source, standards-based, real-time, high-performance CORBA ORB
- Runs on POSIX, Win32, & RTOS platforms
  - *e.g.*, VxWorks, LynxOS, Chorus
- Leverages ACE

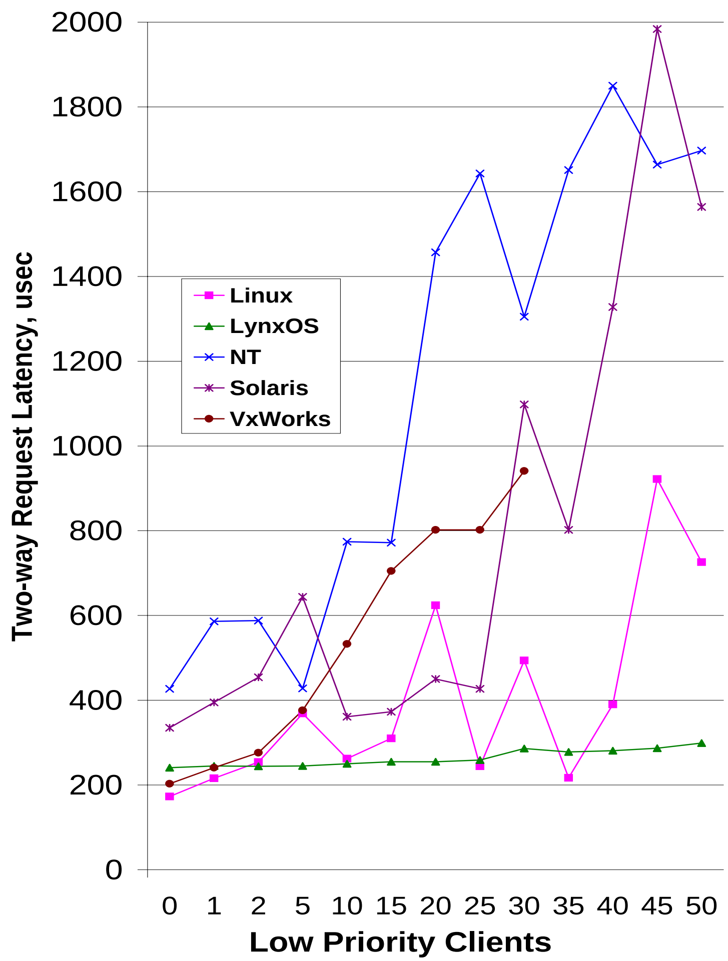
Performance Experiment



`http://www.cs.wustl.edu/~levine/research/RT-OS.ps.gz`

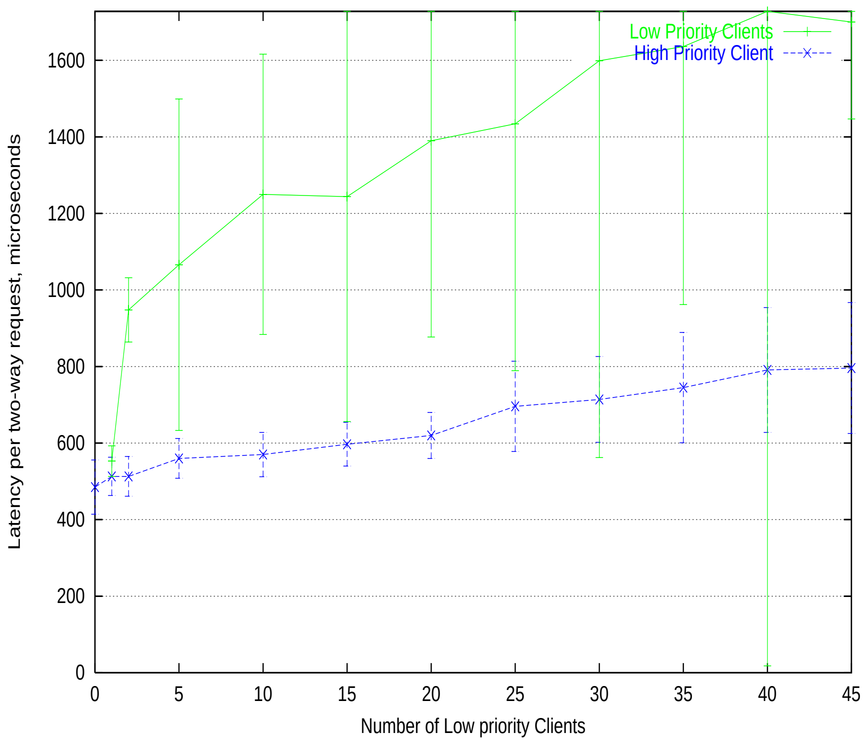
- One 20 Hz high-priority client
- *1..n* 10 Hz low-priority clients
  - Increasing *n* increases load
- Server factory implements *thread-per-connection*
  - Each connection links client with its servant

High-Priority Request Latency Results

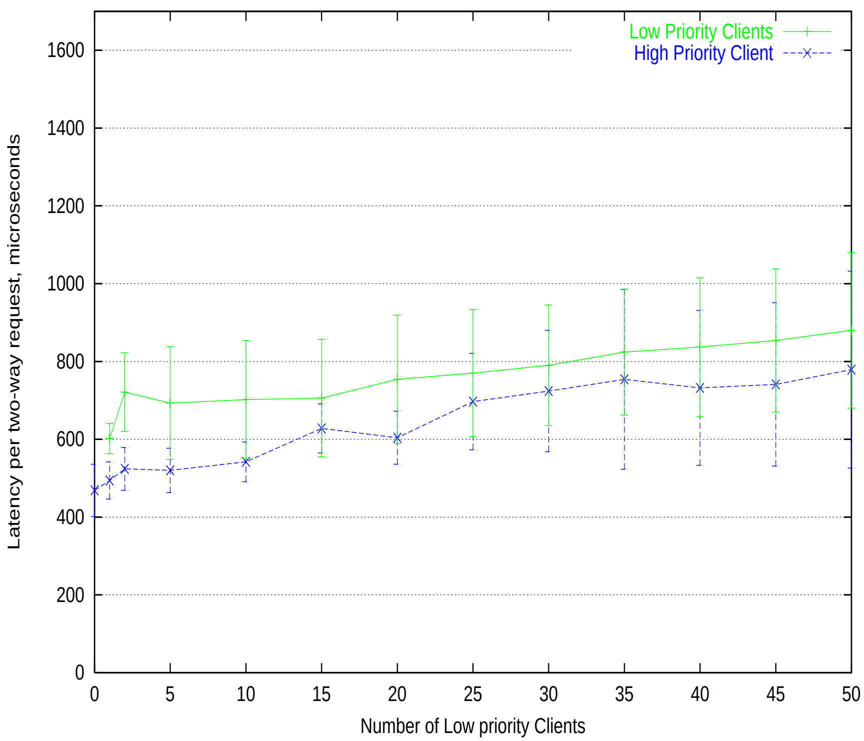


- **Synopsis of results**
  - LynxOS provides consistently low and predictable latency
  - VxWorks does not scale on x86
  - Non-RTOS's are not predictable
  - ORB (TAO) provides low latency and avoids priority inversion
    - *i.e.*, high priority client always has lowest latency

TAO Performance on LynxOS 3.0.0



Server and Client on Same CPU



Server and Client on Different CPUs

Limitations of Current Middleware

- Large footprint
  - Over 2 Mb of code for ACE+TAO libraries
- Lack of end-to-end QoS support
- Configurability does not extend to the code

Limited Configurability

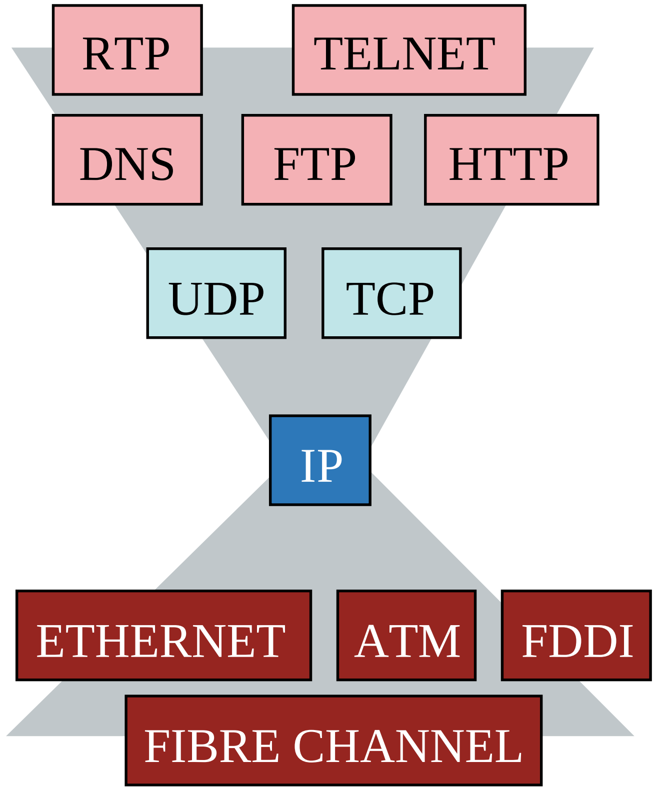
- Mobile applications must be both statically and dynamically configurable.
- Improved static configuration support must be engineered in to avoid linking in all potential code.
- Dynamic linking aids configurability, but:
  - must be careful to avoid objectionable overhead and unpredictability.
  - raises security issues.

Next Generation Middleware Features

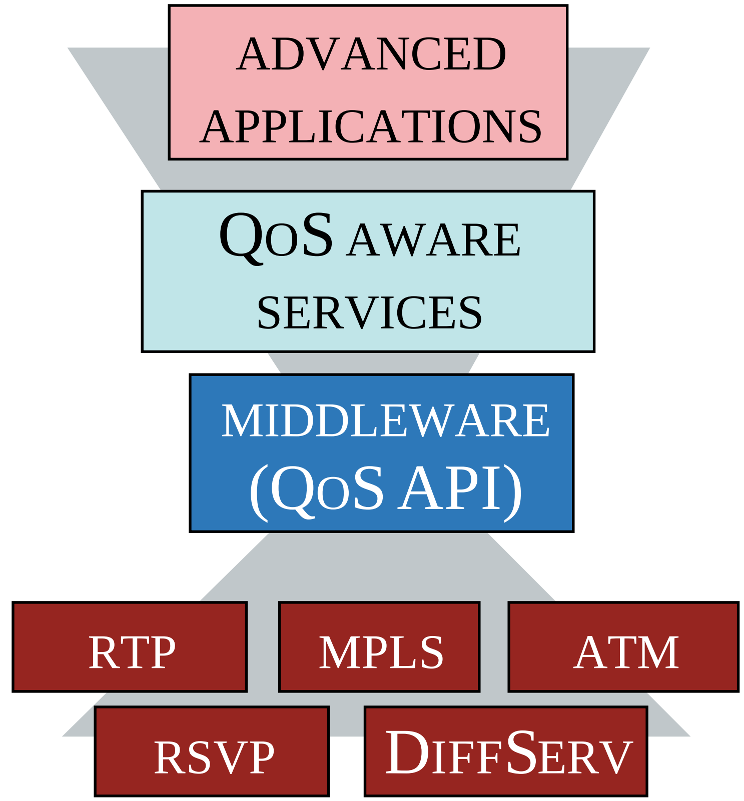
Next generation middleware must have:

- Smaller footprint
  - Middleware tries to provide for potential needs
  - It's difficult to include *only* the middleware code that a particular application needs
  - Demand will drive static footprint down
- Standardized real-time support
- Native QoS support
- Better configurability
- Better mobility support
- Smaller footprint

QoS Enabled Middleware

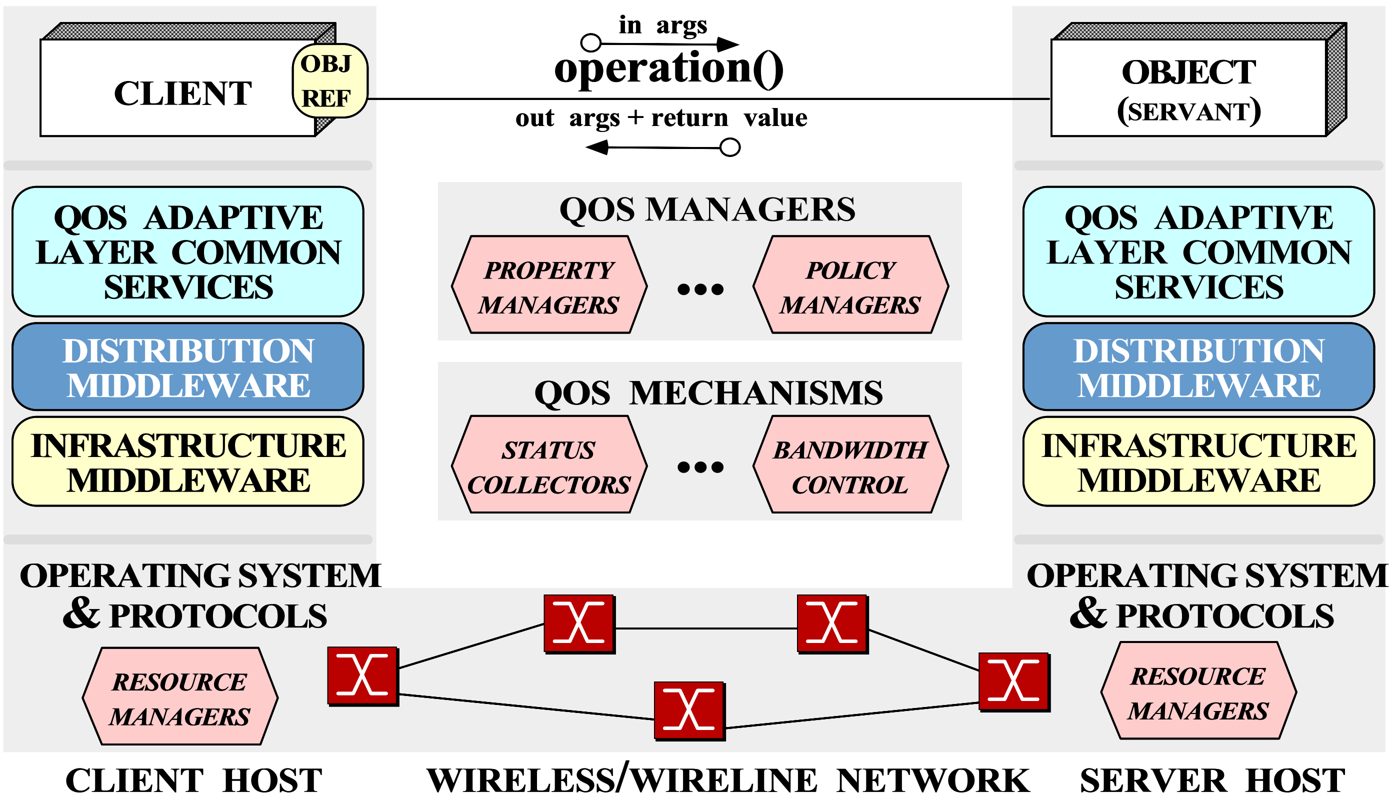


INTERNETWORKING ARCH



QOS ENABLED MIDDLEWARE ARCH

Creating a Framework to Support QoS Enabled Middleware



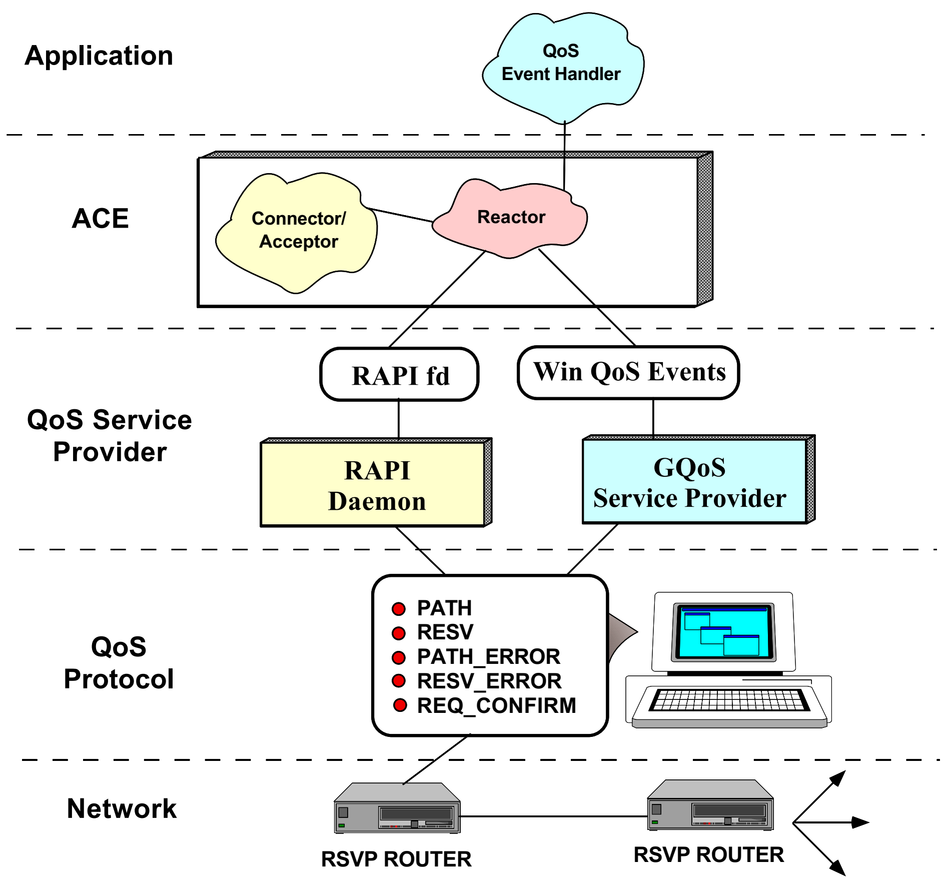
Meeting End-to-End QoS Requirements

- **Design Challenges**
  - QoS requirements specification
    - Two levels of specifying QoS - application (*e.g.*, audio sample, video frame rate) and network (*e.g.*, service type, bandwidth) levels.
  - Meeting operation scheduling deadlines
  - Alleviating priority inversion and non-determinism
  - Reducing demultiplexing overhead

ACE QoS API Overview

- Unified view of different QoS technologies
- Portability, QoS Parameters, Extensibility.
- Wrappers for low level QoS APIs
- Notion of a QoS session
- Handling QoS events through the ACE Reactor
- Limitations because of generalization

Event Notification



QoS Mapping

- Translation of QoS specifications between different levels, *e.g.*, between application and network levels.
- Reserve network resources at connection establishment.
- Good mapping rules to avoid reservation of too much (or too little) resources.
- QoS specification and parameter mapping.
- Required both at connection establishment and renegotiation time.

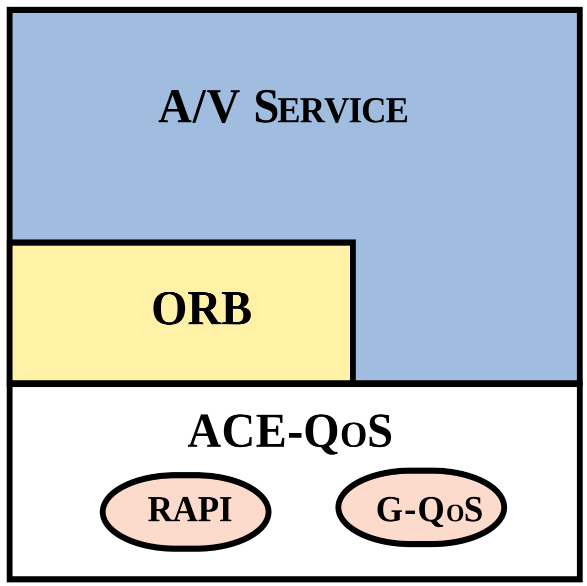
QoS Monitoring and Adaptation

- **QoS Monitoring**
  - Mechanism for measuring end-to-end QoS parameters over a finite time period.
  - Typically done on the receiving side.
  - Notification of QoS changes and violations to the application through feedback channels.

- **QoS Adaptation**
  - Take actions based on the measured QoS and the application QoS requirements.
  - Typically done on the sending side.
  - Adaptation can be at the transport (*e.g.*, flow control), application (*e.g.*, MPEG-II coding rate adaptation) and at the signalling (*e.g.*, QoS renegotiation) levels.

QoS-Based Transport API

- Provides calls for provisioning, control (renegotiation and violation notification) and media transfer.
- ACE-QoS API's provide the required QoS-based transport API.



Mobility Middleware Design Guidelines

- Middleware can reduce lifecycle cost and time-to-market
- Mobile imposes additional constraints on middleware
- Next generation middleware must:
  - Have smaller footprint
  - Provide QoS support, including dependability. For example:
    * Define generic QoS mappings for various flows.
    * Design a flexible and extensible QoS monitoring and adaptation framework.
    * Understand QoS specifications for different flow protocols.
  - Support configurability
  - Support adaptability

For Further Information

- More detail on TAO:
  `http://www.cs.wustl.edu/~schmidt/RT-ORB.ps.gz`
- TAO Event Channel:
  `http://www.cs.wustl.edu/~levine/research/JSAC98.ps.gz`
- ORB Endsystem Architecture:
  `http://www.cs.wustl.edu/~schmidt/RT-middleware.ps.gz`
- OS Comparison:
  `http://www.cs.wustl.edu/~levine/RT-OS.ps.gz`
- These slides:
  `http://www.cs.wustl.edu/~levine/research/srs00.ps.gz`

