Quality of Service Design Considerations for NASPInet

Prof. Dave Bakken

Washington State University

NASPI Working Group Meeting
Scottsdale, Arizona, USA
February 4, 2009

Outline

- **Context**
- Quality of Service (QoS) Basics
- QoS Properties, Mechanisms, Resources
- Other Misc. Considerations

NASPInet Service Classes are really mostly QoS Categories!

NASPInet Traffic Attribute	A: Feedback Control	B: Feed-Fwd. Control	C: Post Event	D: Visual-ization	E: Research
Low Latency	4	3	1	2	1
Availability	4	2	3	1	1
Accuracy	4	2	4	1	1 - 4
Time Alignment	4	4	1	2	1 - 4
High message rate	4	2	4	2	1
Path Redundancy	4	4	1	2	1
Key: **4**-critically important **3**-important **2**-somewhat important **1**-not very important					

- NASPInet's **only** *raison d'être* is to serve power apps!
- Classes are QoS Categories, each with many algorithms (& parameterizations & configurations thereof)
- Thought Q: how to support many apps all at once….
- Q: "Why not Sprint (or other teleco) for NASPInet?"

NASPInet Architecture

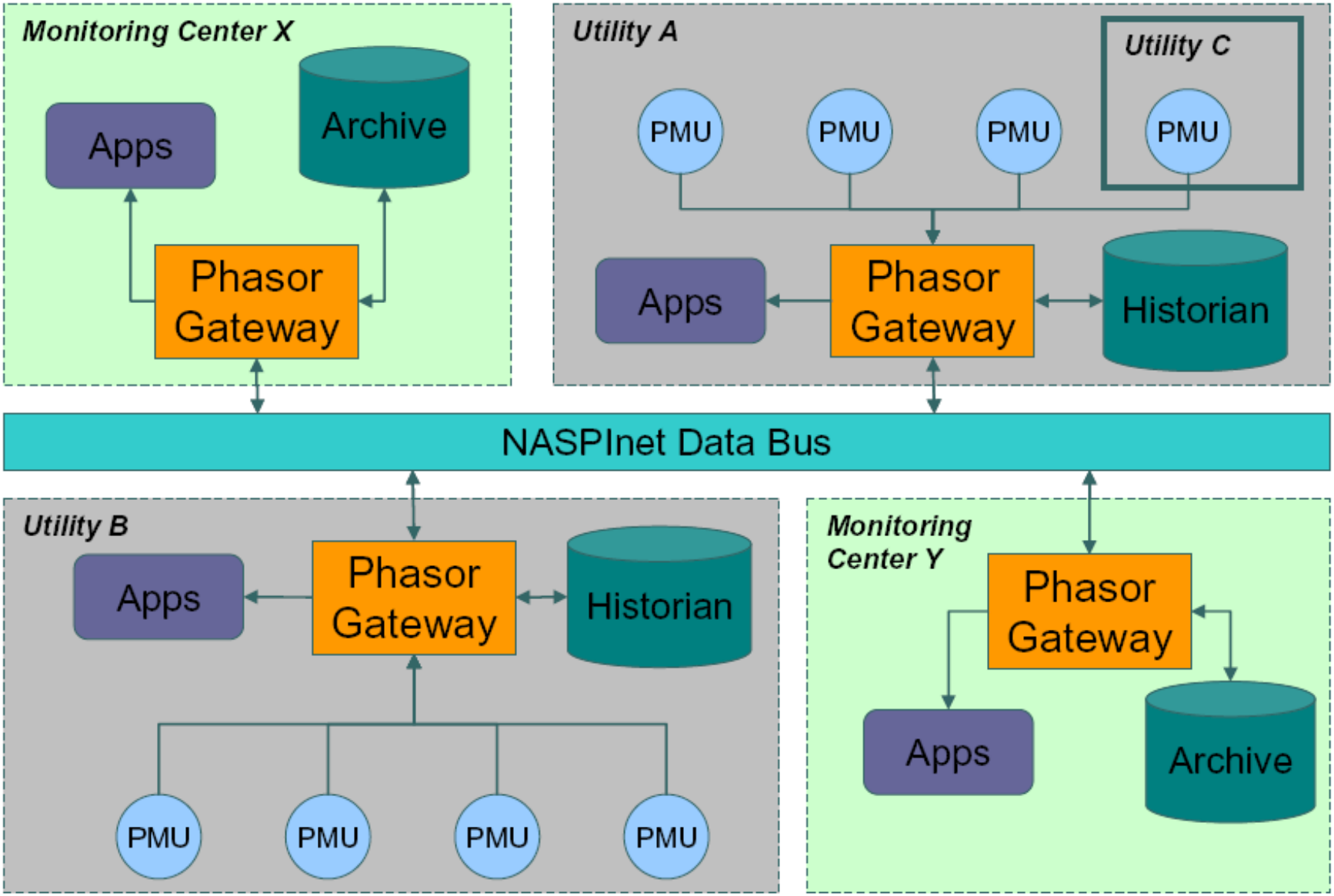
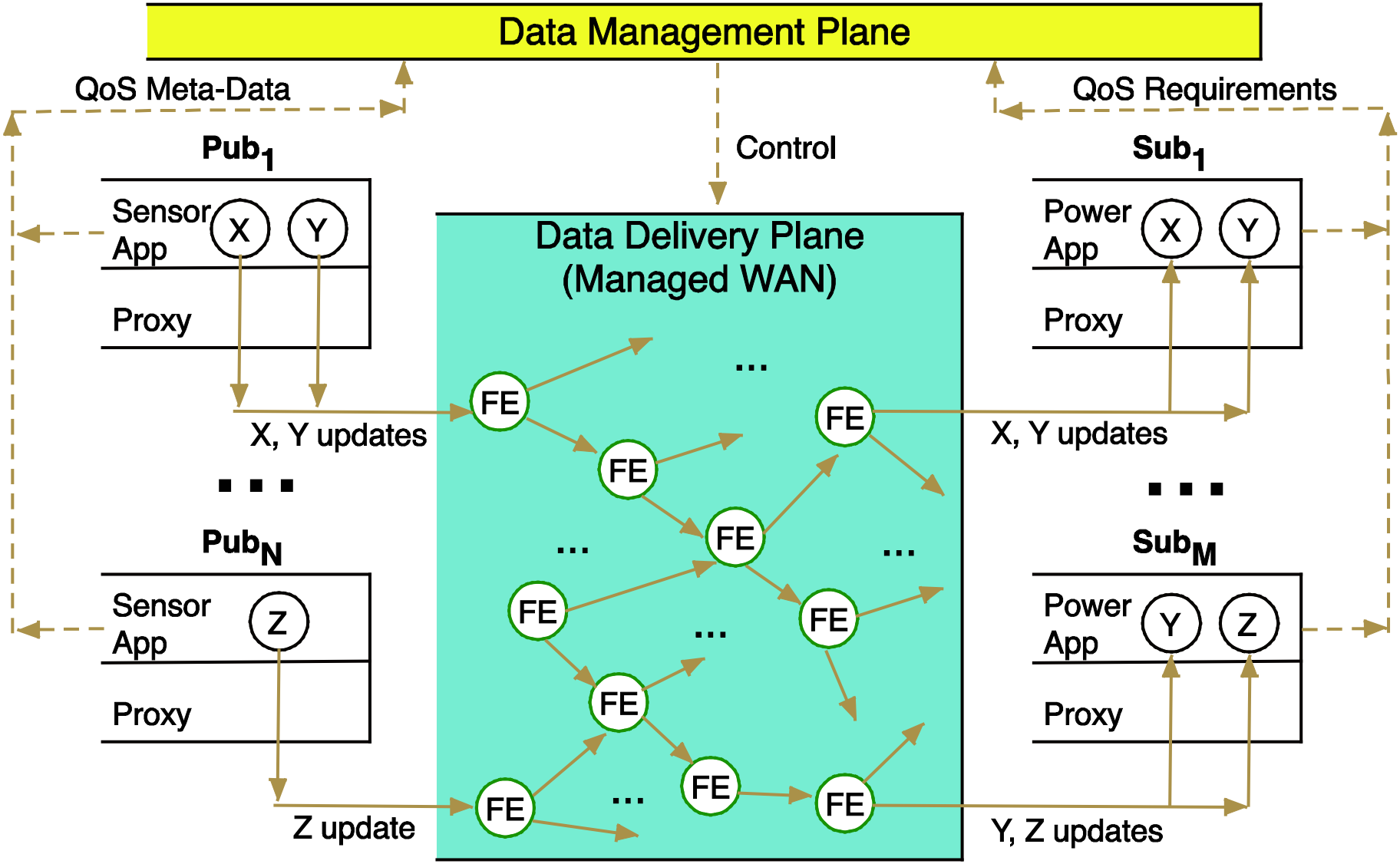


Figure 1 Basic NASPInet Architecture

Most QoS is provided via the Data Bus (GridStat is an instance) from GW↔GW

NASPInet Data Bus (Data Delivery Plane)



NASPInet Data Bus vs. Internet

Requirements more stringent but environment more forgiving

- Smaller scale (~10^4 FEs vs ~10^8 routers)
- Strong QoS guarantees vs. best effort (critical infrastructure), quick recovery
  - Do not allow path instabilities that can occur on the Internet
  - Cannot use ACKs/NACKs to "guarantee" delivery like TCP/IP
- Much more control over traffic
- Much more knowledge of IT topology and usage (*a priori*)
- Multicast is the norm, not the exception
- **Note**: these key differences must directly and explicitly be exploited for better QoS
  - More info on this: see a paper we have in review… (email me)

Outline

- Context
- **QoS Basics**
- QoS Properties, Mechanisms, Resources
- Other Misc. Considerations

What is QoS?

- Usual software API (e.g., CORBA IDL) tells "**what**" can or should be done

  Note: AKA "business logic" or "functional properties"
- Quality of Service is the non-functional "**how**" to do the above "**what**"
- I.e., with how much performance, robustness, cyber-security, quality of result (goodness, correctness), cost, ….
- Note:
  - Older view of QoS: link-level performance
  - More recent view (last decade) is above: **end-to-end**, **multi-dimensional** (supporting multiple QoS properties)

QoS Properties/Dimensions (1)

- **Latency (IEEE 1646, substation scope)**
  - 4 ms within substation, 8-12 external for all but very fastest

  Low latency over hundreds of miles opens up new protection …..
- **Rate** (1/minute to 250/second)
- **Availability of Data** (EPRI IntelliGrid 2004)

Level	*Availibility (%)*	*Downtime/Year*
Ultra	*99.9999*	*~ ½ second*
Extremely	*99.999*	*~5 minutes*
Very	*99.99*	*~1 hour*
High	*99.9*	*~9 hours*
Medium	*99.0*	*~3.5 days*

- Delivered QoS **must** be tailorable per data item & changeable (in SW)

QoS Properties/Dimensions (2)

Other ones are mostly traditional cyber-security…

- **C**onfidentiality
- **I**ntegrity
- **A**vailability (often most important); covered earlier
- Sometimes also: non-repudiation, auditing, …

Also **trust management** (backup slides…)

QoS is "Above the Network" (End-to-End)

- Computer networking techniques necessary but not sufficient
- Need also "**distributed computing**" and **its middleware** to capture end-to-end application requirements (backup slides)
  - Sensor-to-app (QoS instrumentation/verification: end-to-end)
  - Higher-level adaptations (than link-level or transport level)
- Keep it manageable:
  - Keep business logic separate from QoS parameters & adaptations
  - I.e., "separation of concerns", don't entangle above

Outline

- Context
- QoS Basics
- **QoS Properties, Mechanisms, Resources**
- Other Misc. Considerations

Providing QoS Properties (1)

- "You can't have it all": **can <u>not</u>**
  - Provide max value of all dimensions/properties
  - Provide precise levels of all properties at once (or even most)
- QoS **<u>allocation</u>** (AKA **QoS Management**) maps from QoS requirements (properties) onto mechanisms/algorithms/protocols that provide it
  - Different mechanisms provide different levels of QoS and take different levels of **resources** (CPU, bandwidth, storage)
  - Allocation done at runtime (connection setup usually), need to have QoS provisioning/planning so there are enough resources

Providing QoS Properties (2): Mechanisms

- **Latency** mechanisms: chain of
  - Network-level QoS "reservations" for performance
  - Real-time bounds in NASPInet FEs
- **Confidentiality** mechanisms: encryption
- **Integrity** mechanisms: higher-level algorithms built on top of encryption (e.g., digital signing)
- **Availability** mechanisms: replication (spatial, temporal, value) & end-to-end latency guarantees (below)

Providing QoS Properties (3): Mapping Down

Higher-level required QoS properties **mapped onto** lower-level properties and then onto available mechanisms

- Appl-level-1: ***freshness*** = *max_period* + *max_latency*
- Appl-level-2: ***rate*** to delivery a given update over given path of links (each with given link-level latencies)
- Network-level-1: ***bits/second*** over a given link
- Network-level-2: ***parameters*** of given network-level QoS ***mechanism***
  - INTSERV/RSVP: buffers in routers along the path
  - DIFFSERV: service class/priority

Must keep the app-level requirements as high as possible

- Will change
- Different mechanisms available in different configurations

Outline

- Context
- QoS Basics
- QoS Properties, Mechanisms, Resources
- **Other Misc. Considerations**

Aperiodic Events

- So far dealing with periodic events that are very predictable
- Have to handle aperiodic events whose rates cannot always be known *a priori*:
  1. Power control messages (to actuator or control center)
  2. Power alarms/alerts
  3. IT control messages (configure or adapt NASPInet)
  4. IT instrumentation messages (performance, intrusion detection)
  5. IT alarms (e.g., end-to-end QoS violation detected at subscriber)
  6. Service classes C (Post-Event) & E (Research); ?DFRs
- Strategy for handling aperiodic events
  - Allocate each kind above a fixed % of resources & enforce in HW
  - Intelligent aggregation of #2 and #5 (the unpredictable ones)
  - Other lower-level techniques in real-time computing R&D

Keeping it Evolvable over Lifecycle ($$$$)

- Don't hard-code required QoS properties into power apps
  - Will change over life-cycle with more studies (!recompile code)
  - Will change at runtime under different operating regimes – steady state vs. cyber-attack vs. power contingency "drilling down" …)

  Alternative: separate requirements for IT integrator/operator
- Don't hard-code available (or assumed) mechanisms or resources (e.g., Navy ships)
  - Will change over lifecycle with system growth
  - Will change at runtime with IT failures and cyber-attacks

  Alternative: map properties onto existing mechanisms; use policies for resource allocation constraints(next slide)
- Note: I see **a lot** of this hard-coding in the power industry
  - E.g., latch onto bridged ethernet or IPv6 multicast

  Alternative: focus on required properties, not mechanisms+resources

Resource Policies for Flexibility

- Policies can be simple database or simple language, about QoS or resources (not just cyber-security)
- Examples:
  - "Which entities/utilities get access permission for what sensor variables under what conditions ?"
  - "How much bandwidth through my domain should I allow utility X to have for power application Y or ancillary service Z, and under what conditions"?
  - "How are the (runtime) conditions above defined and measured?"
  - ….
- Adaptive strategies are really a kind of policy (and involve tradeoffs) that provide a higher-level mechanism

Q: What QoS Does Your NASPI App Need?

- What max latency & rate (freshness) do you really need?
- How can your application suffer the following and still do what it needs to:
  - Intermittently longer latencies
  - Intermittent short spurts of dropped messages
  - Outages of some of its input data
  - …

Not studied much by power community, but need to IMO!

Conclusions

- QoS is
  - the “how” a service does “what” it does
  - multi-dimensional
  - Not just above the traditional “network level” (e.g., Cisco routers) but also more end-to-end (distributed computing & middleware)
- NASPInet requires guaranteeing a wide range of multi-dimensional QoS combinations
  - Can leverage lessons learned from military systems
  - But exploit “better” characteristics (smaller scale, more control, ...)
  - NASPInet will be complex middleware ➔ needs to apply a lot of distributed computing knowledge

“You have an IT Problem, not a power problem”

- *Matt Heere, OSISoft, September 2007*

For More Info (1)

- Zinky, John A. and Bakken, David E. and Schantz, Richard E., "Architectural Support for Quality of Service for CORBA Objects", *Theory and Practice of Object Systems* (Special Issue on CORBA and the OMG), 3:1, April 1997, 55–73.
  - What QoS issues applications face when they have to run over a wide area
  - How QoS gets entangled in the business logic and the problems it causes
  - What systematically can be done about the above (Multi-dimensional QoS middleware, separation of concerns, …)
  - Version 1 of Quality Objects (QoO) middleware
  - (Heavily cited paper)
  - http://www.dist-systems.bbn.com/papers/1997/TAPOS/

For More Info (2)

- David E. Bakken, Carl H. Hauser, Harald Gjermundrød, and Anjan Bose. "Towards More Flexible and Robust Data Delivery for Monitoring and Control of the Electric Power Grid", *Technical Report EECS-GS-009*, School of Electrical Engineering and Computer Science, Washington State University, May 30, 2007.
  - How distributed computing can help NASPInet
  - GridStat overview
  - http://www.gridstat.net/TR-GS-009.pdf

For More Info (3)

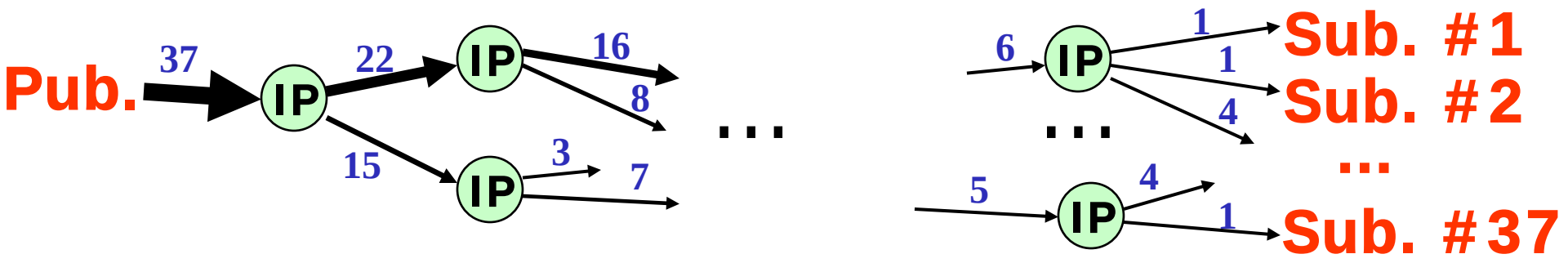
- David Bakken, Carl Hauser, Harald Gjermundrød. "Periodically Updated Variables: Wide-Area Publish-Subscribe Middleware Supporting Electric Power Monitoring and Control", submitted for review to the 2009 IEEE International Conference on Distributed Computing Systems (ICDCS).
  - How NASPInet environment is different from the Internet
  - Much more precise definition of NASPInet data bus requirements for streaming real-time data
  - Email Dave Bakken for a copy
- Contact coordinates: bakken@wsu.edu 509-335-2399 www.gridstat.net

Outline

- Context
- QoS Basics
- QoS Properties, Mechanisms, Resources
- Other Misc. Considerations
- **Backup Slides**
  - **Issues on QoS Properites & Mechanisms & Resources**
  - Middleware and Distributed Computing
  - Smart Grids for Distribution need good communication & synchronized data
  - Other Misc. Electricity+IT slides
  - Some GridStat Details

A Crucial Note on Network & Publisher Load

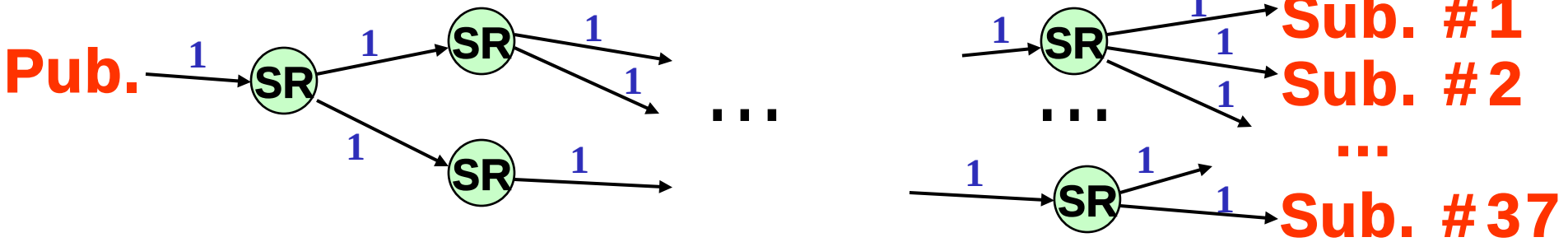
Direct network programming: up to **37 copies** (**#subs**) of a given update:



Problems: (1) network load (2) publisher load (3) multiple encrypts

Note on IP Multicast (1) not everywhere (2) can't do per-sub QoS

GridStat/NASPInet: **1 copy** (max) of a given update on any network link:



Note: per-subscriber QoS (rate, lantency, #paths) via rate filtering: if a subscriber (or subtree) does not need a given update it is not sent on

QoS Properties/Dimensions (2) TRUST

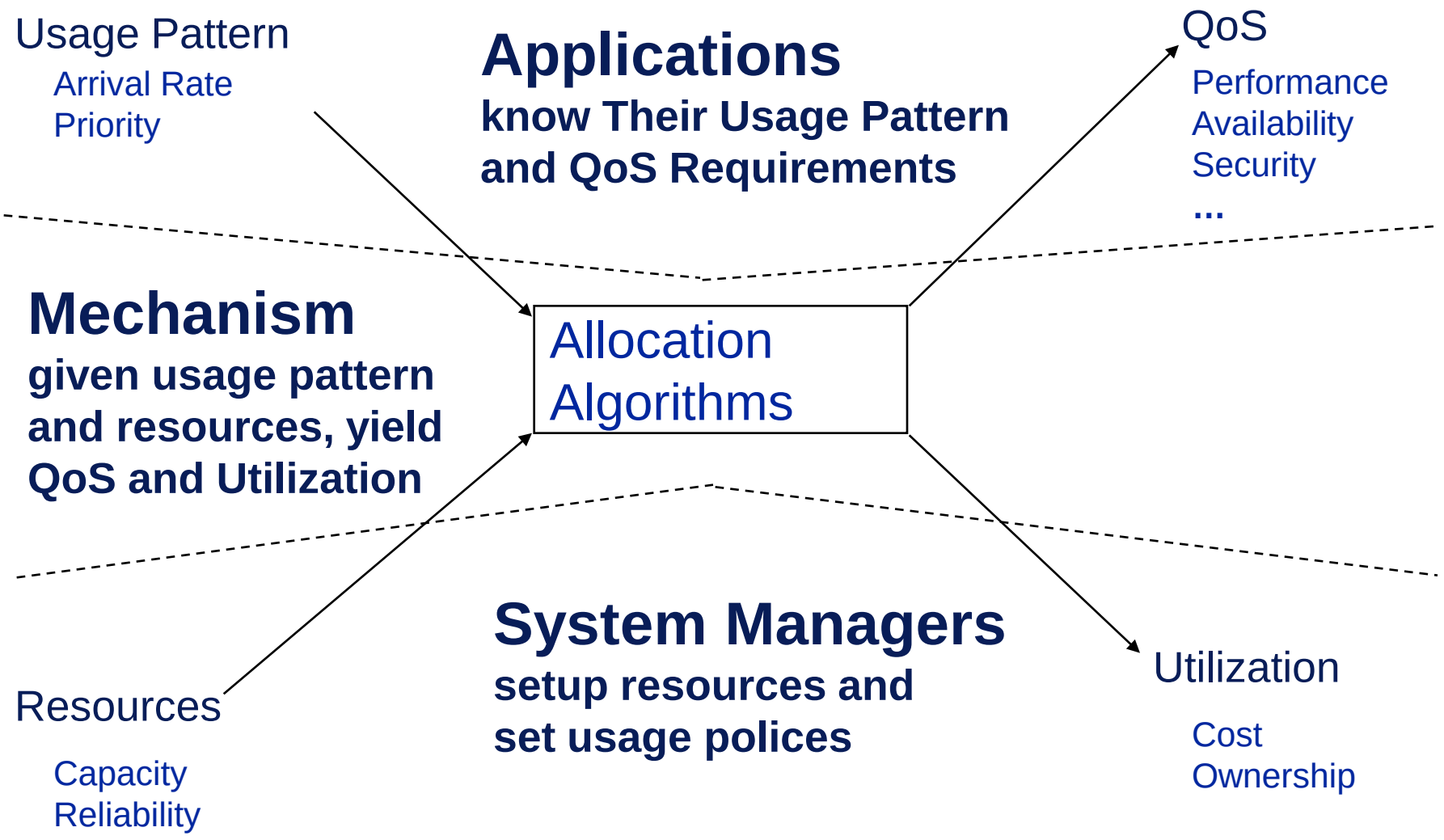
Other ones are mostly traditional cyber-security…

- **C**onfidentiality
- **I**ntegrity
- **A**vailability (often most important); covered earlier
- Sometimes also: non-repudiation, auditing, …

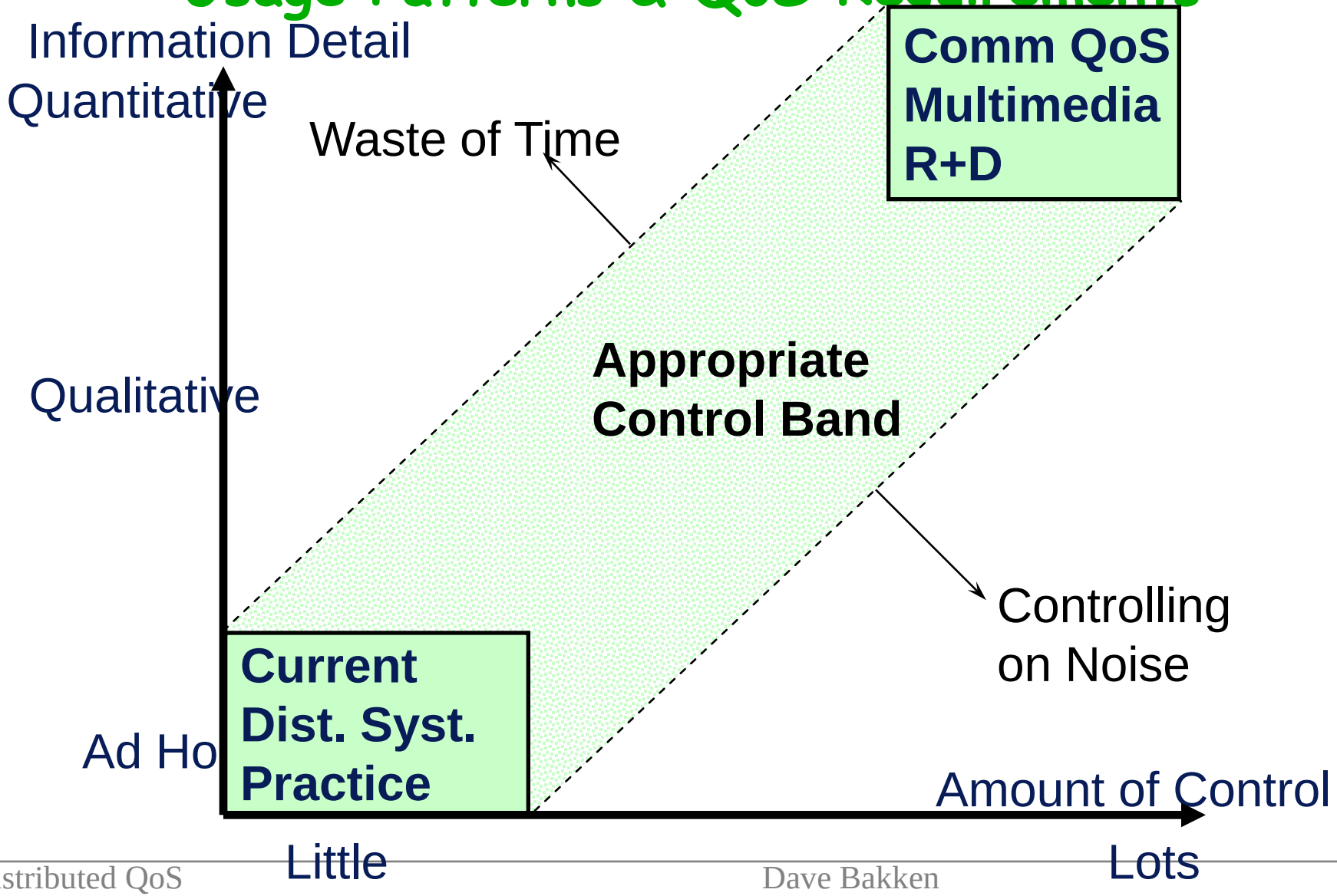
But traditional cyber-security is not enough

- **Trust management**: systematically reasoning about
  - How much trust to place in data received, especially when via
    - Chains of processing (pub-sub)
    - Aggregation of many different inputs from different sources
  - How much access to data to provide potentially untrustworthy entities
  - How to certifiably introduce an outside entity for access control

QoS-Aware Resource Management I: Many Mechanisms Give the Correct Functionality, But Are Appropriate for a Small Set of Conditions



QoS-Aware Resource Management II: Control over Resource Allocation is Useless w/o Information on Usage Patterns & QoS Requirements



Application-Level Adaptation Choices

- Q: How can distributed applications become more predictable and adapt to changing system conditions?
  - Control and Reserve Resources
  - Utilize alternate Resources (redundancy)
  - Use an alternate mechanism (with different system properties)
  - Take longer
    - reschedule for later
    - tolerate finishing later than originally expected
  - Do less (Happiness ≡ success / expectations)
- Note the multiple possible layers of adaptation (a) Client application logic (b) Above the middleware core on client-side (c) Inside the middleware (d) Above the middleware core on server-side (e) Server logic
- Premise: supporting all the above choices is helpful!

Networking Review

- Networking technology radically improved in last decade
- Some network technologies
  - Internet Protocol (IP): links LANs
    - best-effort (no QoS)
    - IPv6 has multicast (unreliable; can't do rate filtering & per-sub QoS)
  - TCP: retransmits data to provide reliability; no predictable latency
  - ATM: packet-switching over virtual circuits
    - Bandwidth guarantees, good latency predictability and reliability (QoS)
    - No multicast
- Bottom line: you can't just "plug in a network" using only off-the-shelf technology for next-generation power grid communications (see TR 009 for detailed explanation)
  - IPv6 & ATM provide useful building blocks, but more is needed
  - Need **managed data delivery services** **above network level**

Outline

- Context
- QoS Basics
- QoS Properties, Mechanisms, Resources
- Other Misc. Considerations
- **Backup Slides**
  - Issues on QoS Properites & Mechanisms & Resources
  - **Middleware and Distributed Computing**
  - Smart Grids for Distribution need good communication & synchronized data
  - Other Misc. Electricity+IT slides
  - Some GridStat Details

What is Distributed Computing?

- Computer networking: gets bytes from A to B
- **Distributed Computing**: answers the following kinds of questions on ***how*** to use a network for given purposes:
  - How to synchronize and replicate data
  - How to structure the architecture of large distributed systems
  - How to handle end-to-end, application-level: fault tolerance, security, quality of service (QoS)
  - How to create **middleware** (esp. for wide area) that makes applications easier to program, manage at runtime, evolve over lifetime, …

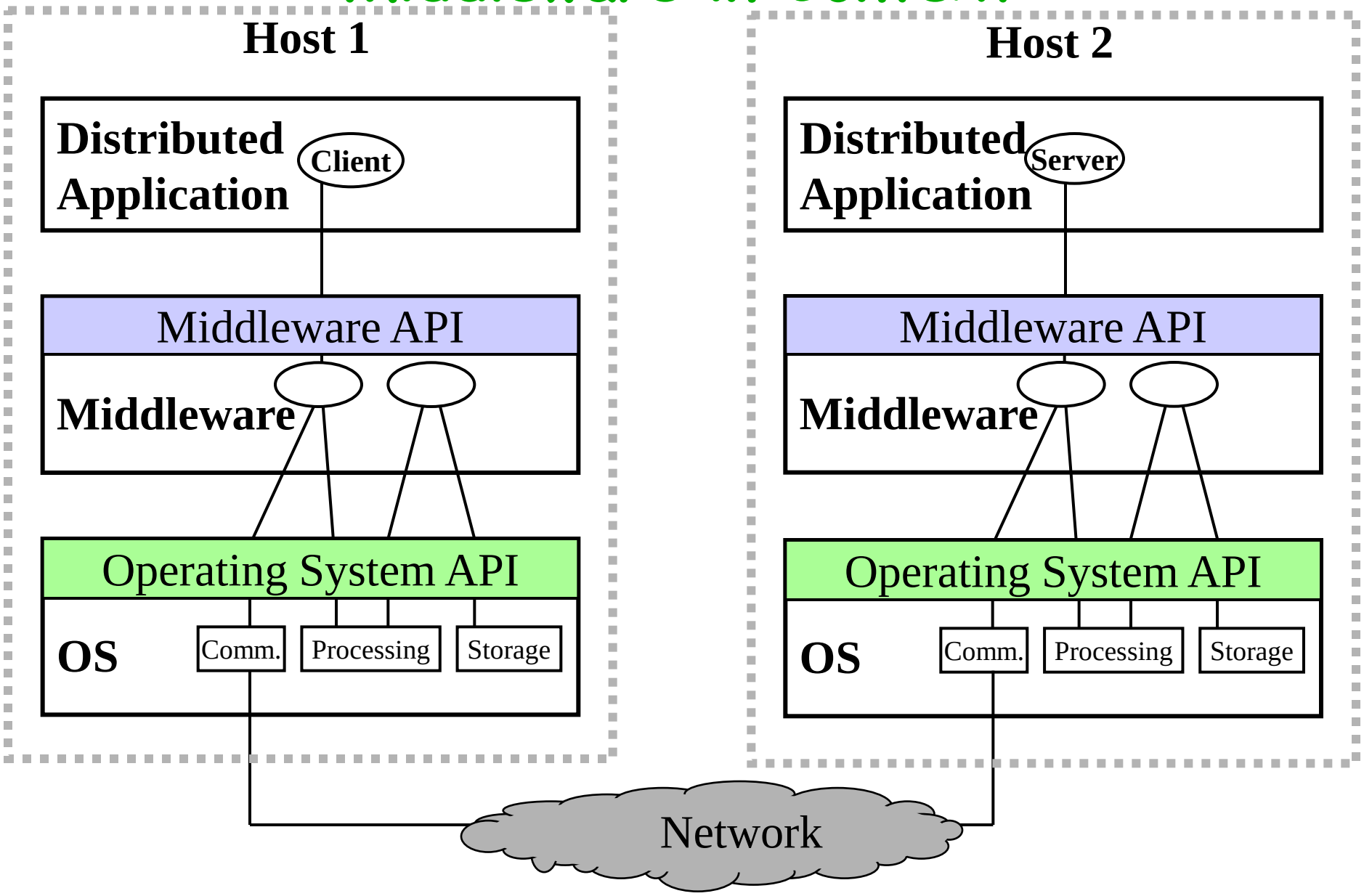
Context: (Most) Technology Marches On

- Hardware technology's progress phenomenal in last few decades
  - Moore's Law
  - Metcalf's Law
  - Graphics processing power
- Software technology's progress is much more spotty
  - "Software crisis"
  - Yet SW is a large and increasing part of complex apps/systems!
- Apps and systems are rapidly becoming (more) networked
  - Oops, distributed software is much harder yet to get right…
- Middleware a promising technology for programability of distributed systems
  - Also fertile grounds for adaptivity and dependability….

Why Middleware?

- Middleware == "**A layer of software above the operating system but below the application program that provides a common programming abstraction across a distributed system**"
- Middleware exists to help manage the complexity and heterogeneity inherent in distributed systems
- Middleware provides higher-level building blocks ("abstractions") for programmers than the OS provides
  - Can make code much more portable
  - Can make them much more productive
  - Can make the resulting code have fewer errors
  - Analogy — MW:sockets ≈ HOL:assembler
- Middleware sometimes is informally called "plumbing"
  - Connects parts of a distributed application with "data pipes" and passes data between them

Middleware in Context



Middleware Benefit: Masking Heterogeneity

- Middleware's programming building blocks mask heterogeneity
  - Makes programmer's life much easier!!
- Kinds of heterogeneity masked by middleware (MW) frameworks
  - All MW masks heterogeneity in network technology
  - All MW masks heterogeneity in host CPU
  - Almost all MW masks heterogeneity in operating system (or family thereof)
    - Notable exception: Microsoft middleware (*de facto*; not *de jure* or *de fiat*)
  - Almost all MW masks heterogeneity in programming language
    - Noteable exception: Java RMI
  - Some MW masks heterogeneity in vendor implementations
    - CORBA best here

Middleware Benefit: Transparency

- Middleware can provide useful transparencies:
  - Access Transparency
  - Location transparency
  - Concurrency transparency
  - Replication transparency
  - Failure transparency
  - Mobility transparency
- Masking heterogeneity and providing transparency makes programming distributed systems much easier to do!

Programming with Middleware

- Programming with Middleware
  - Do not have to learn a new programming language! (Usually)
  - Use an existing one already familiar with: C++, Java, C#, Ada, (yuk) Visual Basic, (yuk) COBOL
- Ways to Program with Middleware
  1. Middleware system provides library of functions (Linda, others)
  2. Support directly in language from beginning (Java and JVM)
  3. External Interface Definition Language (IDL) that "maps" to the language and generates local "proxy"

Kinds of Middleware

- Distributed Tuples: (a, b, c, d, e)
  - Relational databases, SQL, relational algebra
  - Linda and tuple spaces
  - JavaSpaces (used by Java Jini)
- Remote procedure call (RPC)
  - make a function call look local even if non-local
- Message-Oriented Middleware (MOM)
  - messages and message queues
- Distributed Object Middleware
  - Make an object method look local even if non-local
  - CORBA
  - DCOM/SOAP/.NET
  - Java RMI

Kinds of Middlware (cont.)

Different middleware systems encapsulate and integrate the different kinds of resources with varying degrees:

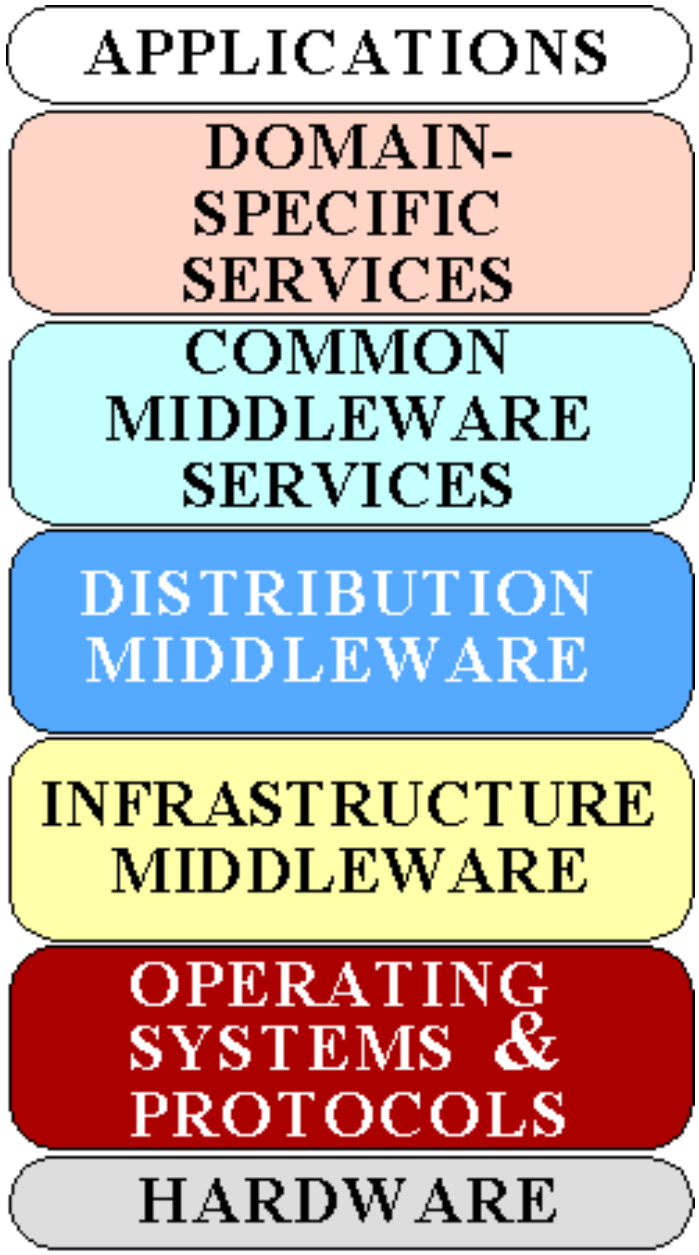
Middleware Category	Communication	Processing	Storage
Distributed Tuples	Yes	Limited	Yes
Remote Procedure Call	Yes	Yes	No
Message-Oriented MW	Yes	No	Limited
Distributed Objects	Yes	Yes	Yes

For many (non-database) applications, and supporting adaptation, distributed object middleware is better because it is more general

Middleware and Legacy Systems

- Legacy systems are a huge problem (and asset) in industry and military domains!
- Middleware often called a "glue" technology: integrated "legacy" components
  - Much distributed programming involves integrating components, not building them from scratch!
- Middleware's abstractions are general enough to allow legacy systems to be "wrapped"
  - Distributed objects are best here because more general
  - End result: a very high-level "lowest common denominator" of interoperability

A Middleware Layering Taxonomy (Schantz)



- Domain-Specific Services
  - Services and APIs tailored to (and reusable only within) certain domains (health care, telecommunications, etc)
  - Examples: CORBA Domain Interfaces, Boeing Bold Stroke architecture
- Common MW Services
  - Adds high-level, domain-independent reusable services for events, fault tolerance, security,
  - Examples: CORBAServices, Eternal
- Distribution MW
  - Provides rich distributed object model that supports much heterogeneity and transparency
  - Examples: CORBA, .NET., Java RMI
- Infrastructure MW
  - Encapsulates core OS Comm. and concurrency services (sometimes enhances them too)
  - Examples: JVM (and other VMs), ACE, group

(Figure courtesy of D. Schmidt)

Outline

- Context
- QoS Basics
- QoS Properties, Mechanisms, Resources
- Other Misc. Considerations
- **Backup Slides**
  - Issues on QoS Properites & Mechanisms & Resources
  - Middleware and Distributed Computing
  - **Smart Grids for Distribution need good communication & synchronized data**
  - Other Misc. Electricity+IT slides
  - Some GridStat Details

Smart Grid needs better data delivery!

The SMART GRID will*:

- Enable active participation by consumers (AMI,DR)
- Accommodate all generation & storage options
- Enable new products, services, & markets
- Provide power quality for the digital economy
- Optimize efficient asset utilization & operation
- Anticipate and respond to system disturbances
- Operate resiliently against attack & natural disaster

Smart Grid applications need systematic and pervasive wide-area data delivery services

*Source: Joe Miller, "**The Smart Grid – How Do We Get There?"** June 26, 2008, **SmartGridNews.com** , The Modern Grid Strategy, a DOE-funded project conducted by the National Energy Technology Laboratory

Distribution-Side Smart Grids SHOULD exploit synchrophasors!

- Voltage instability: compare voltage angles across transmission and distribution
- Distribution substation busbar protection
- Deciding when you can loop a distribution system
- Have no single point of failure & much better distribution substation value error checking
- Solve power quality problems: record event to microsecond so correlated with lightning, transmission switching, customer load switching, …
- Sources:
  - Dr. Ed. Schweitzer, ed@selinc.com
  - Dr. Armando Guzman, Armando_Guzman@selinc.com

Outline

- Context
- QoS Basics
- QoS Properties, Mechanisms, Resources
- Other Misc. Considerations
- **Backup Slides**
  - Issues on QoS Properites & Mechanisms & Resources
  - Middleware and Distributed Computing
  - Smart Grids for Distribution need good communication & synchronized data
  - **Other Misc. Electricity+IT slides**
  - Some GridStat Details

Rationale for Better Communications

- US Electric Power Communications System is aging
  - SCADA is1960s technology
  - Not updated meaningfully (no industry investment)
  - Much star-connected, inflexible, slow, crude SCADA "polling"
  - **Little communication between electric utilities**
- Data collection has increased many fold at substations
  - Faster measurement rates, often time synchronized
  - Communications not there to move this data where needed
  - Even ICCP can't keep up with rate & other requirements

Consequences of Limited Data Exchange

- Much less visibility into system
- Greatly limits control opportunities
- Greatly limits protection opportunities
  - Protection almost always limited to local data (w/in substation)
  - SPS/RAS are too expensive, one-off solution
- Limits reliability: major blackout contributor
- Limits profits (!!)
  - Monitoring systems that can run the system at higher load levels (and of course hence with more profitably) are one of the "exciting new technologies that will be tools for the future".

    Root, C. "The Future Beckons", *IEEE Power & Energy Magazine*, 4(1), January/February 2006, 24–31.

"Best" Practices (!) in Power Industry Today

- Caveat: we all have training+experience in some areas, not all!!!!.....
- Cobble together data communications on a **per-project, piecemeal basis**
- Send all data to all recipients (or single control center) at the highest rate anyone needs it at (what else to do?)
- Use TCP/IP or web services (highly inadvisable per CS R&D)
- Unaware of state of art in distributed computing
- Results: inflexible, not robust, expensive and over-budget
  - Huge part of cost of a specialized protection scheme (SPS)

The Writing on the Wall

- Francis Cleveland, Xanthus consulting (**emphasis** ours)
  - *With the exception of the initial power equipment problems in the August 14, 2003 blackout, the on-going and cascading failures were **almost exclusively due to problems in providing the right information to the right place within the right time**.*
- Clark Gellings, EPRI (**emphasis** ours)
  - *"The **ultimate challenge** in creating the power delivery system of the 21st century is in the development of **a communications infrastructure that allows for universal connectivity**."*
  - *"In order to create this new power delivery system, what is needed is a **national electricity-communications superhighway** that links generation, transmission, substations, consumers, and distribution and delivery controllers."*

Grid Getting Less Stable Each Year…

- Generation and demand keeps outpacing transmission
- New sources of energy to integrate
  - *"The answer, my friend, is blowing in the **wind**?"*
  - The problem is, too!
- Flexible, robust, and secure data delivery services are a **mitigating gap technology**
- *Question for **<u>you</u>**: How long will the industry continue these same old "best practices"*

Status Information & the Grid/CIP

- Changing requirements
  - More general topology and connectivity including multicast
  - New services require wider range of quantity, timeliness, …
    - Situation awareness: phone calls (or FAXes) not adequate!
    - 4-second SCADA cycle moving to 4 times or more per power cycle (>800x more data)
  - Existing hardwired, hierarchical structure does not suffice
    - Status items may be needed at multiple locations with different rates, latencies, & criticality (availability)
  - There is increasing concern for data security
    - From random hackers
    - From dedicated adversaries (terrorists?)
    - From disgruntled insiders

Flexibility Requirements

- Multicast (1→many, efficiently)
- Heterogeneity of communication topologies
- Heterogeneity of delivery latency and delivery rate
- Temporal synchronism of rate filtering
- Heterogeneity of computing resources
- Extensibility to new kinds of computing resources
- Open architecture: easy interoperability across multiple vendors

Tomorrow's applications need this flexibility, too: smart grids, advanced metering infrastructure (enabling demand response), distributed generation (microgrids, renewables), …

Outline

- Context
- QoS Basics
- QoS Properties, Mechanisms, Resources
- Other Misc. Considerations
- **Backup Slides**
  - Issues on QoS Properites & Mechanisms & Resources
  - Middleware and Distributed Computing
  - Smart Grids for Distribution need good communication & synchronized data
  - Other Misc. Electricity+IT slides
  - **Some GridStat Details**

GridStat Approach

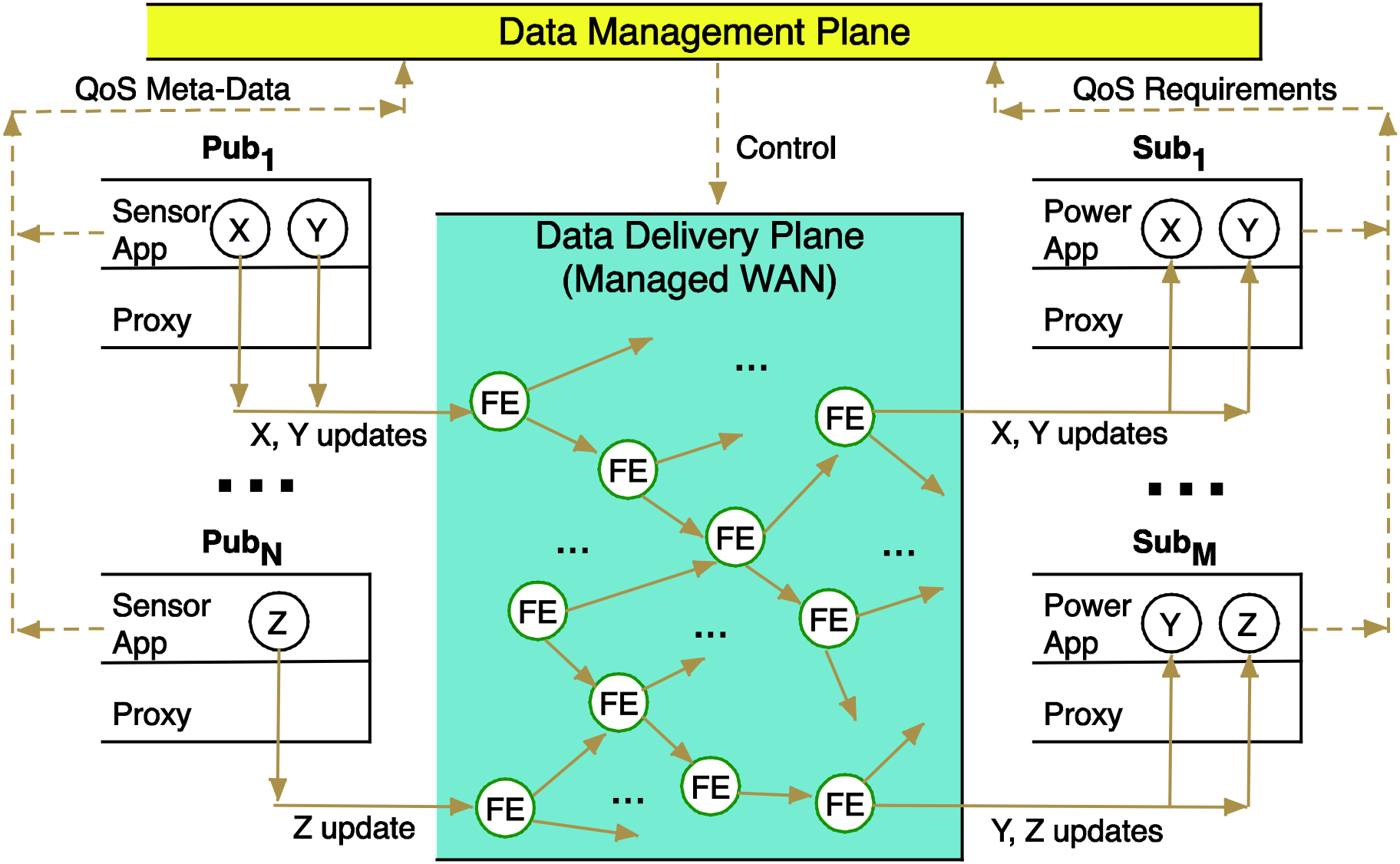
- Build pragmatic, comprehensive end-to-end framework
  - Extensibility & customizability are key (lots of hooks…)
  - Intended to extend to capabilities & scope of large power grid
- "Outside-In" not "Inside-Out"
  - lay down all the end-to-end plumbing, *a la* QuO
- Start with simple QoS & sub-optimal mechanisms
  - Hard QoS guarantees only if we control all access points
  - Provide QoS APIs & hooks to capture requirements to enable many more optimizations and more extensive management
- Extend over time for more coverage of
  - QoS guarantees
  - Adaptability
  - Security

  With more QoS mechanisms, policy languages, validation, ….

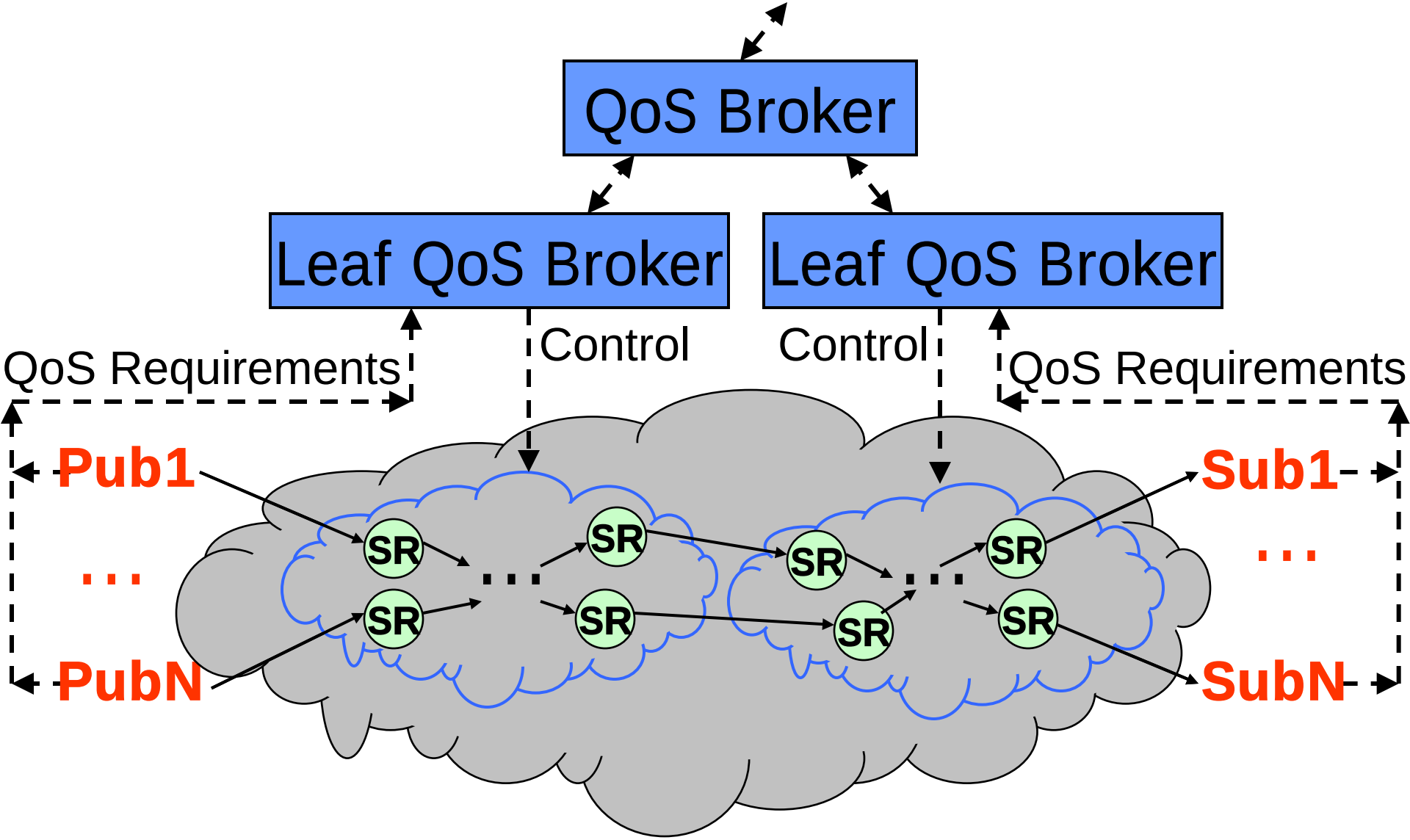
GridStat APIs

- **Pull**
  - A cache instance of the variable kept at each subscribe
  - Subscriber can use just like a local object, when needed
  - Distribution transparency
- **Push**
  - Subscriber can register to get each update
  - Good for database integration (yuk!)
- **QoS Push**
  - Subscriber can register callback to get notified if QoS violated
  - Most apps won't use, but great for aggregation: end-to-end QoS violation

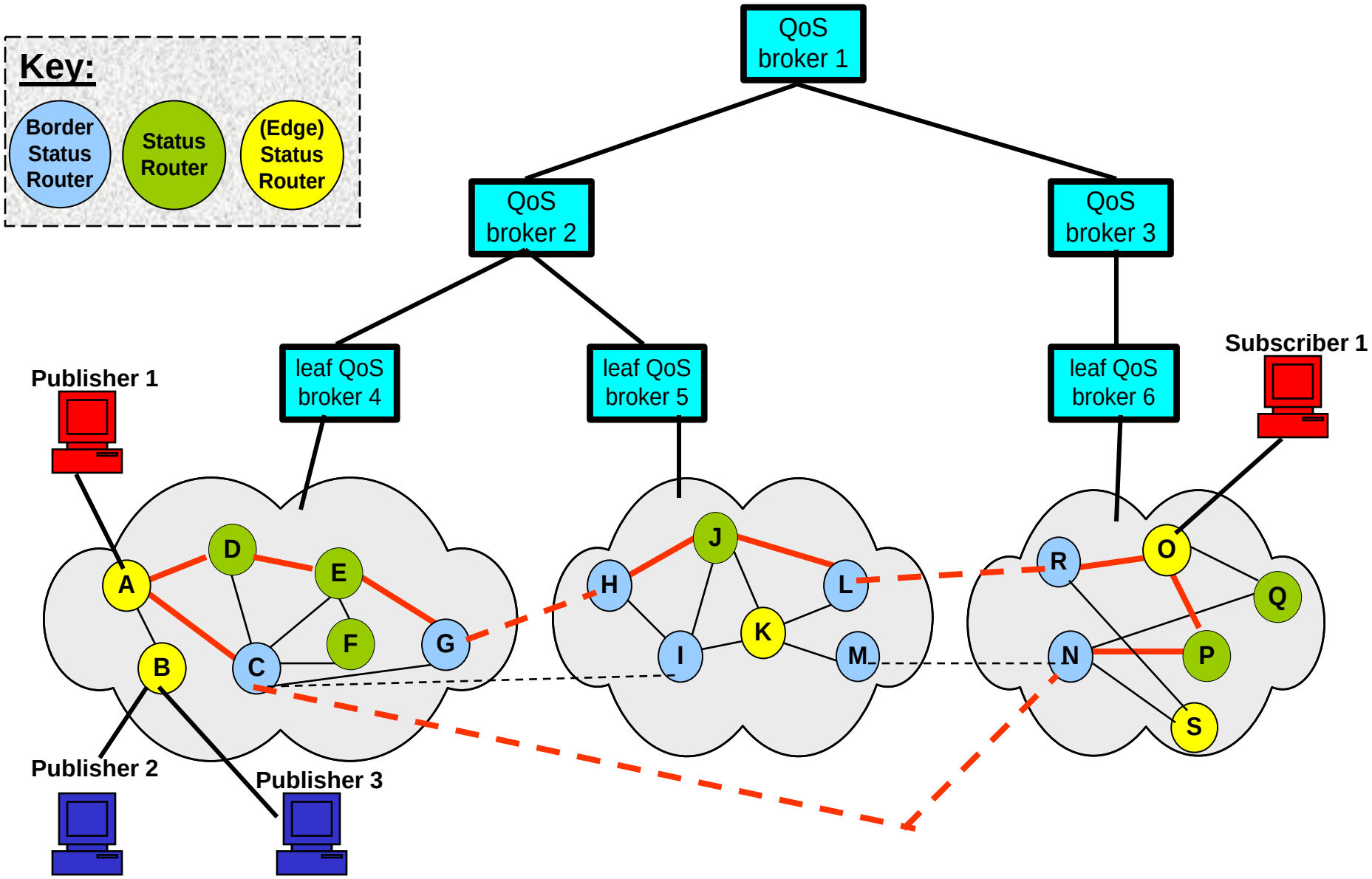
GridStat Architecture



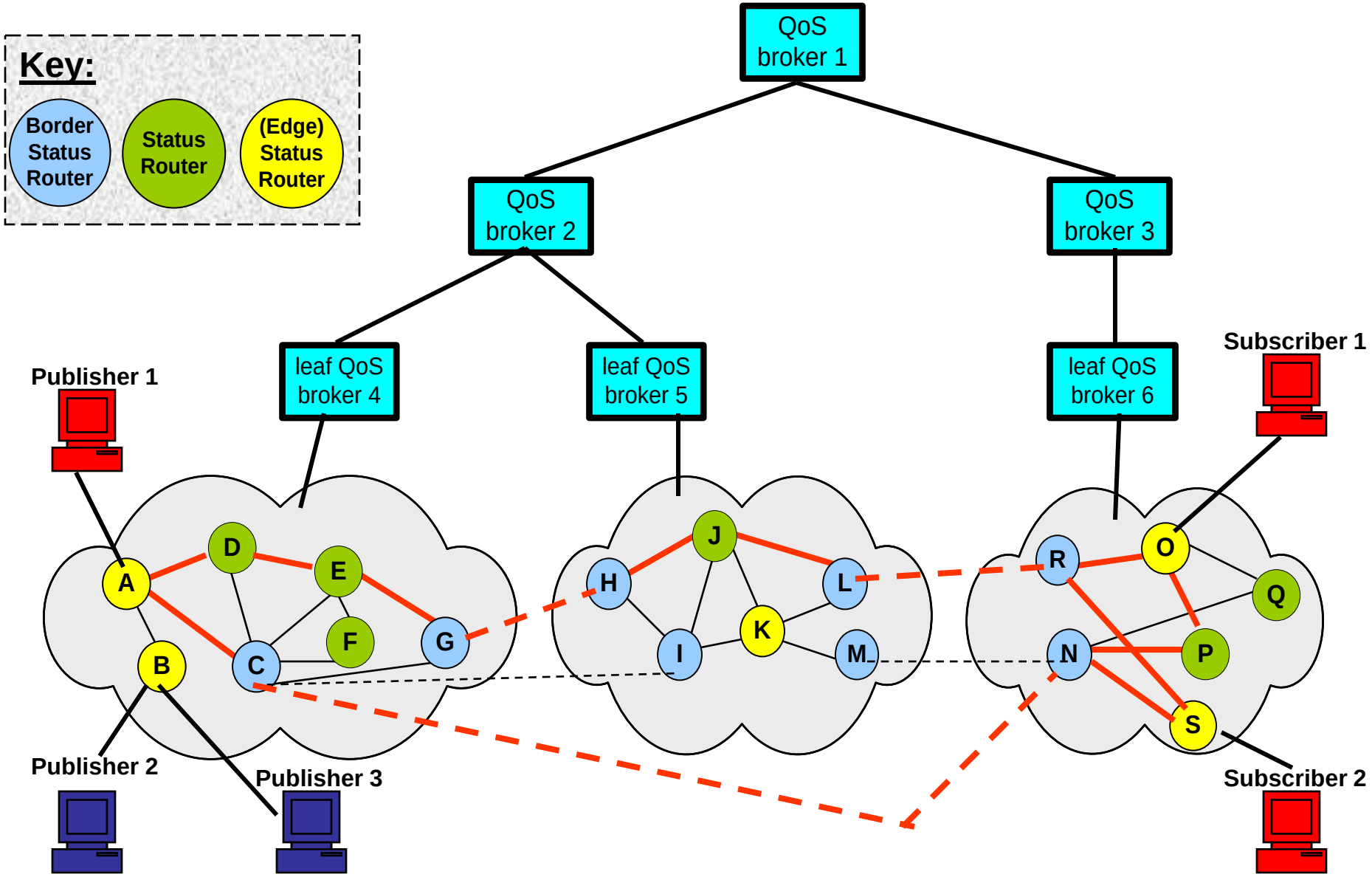
GridStat Architecture



Route Allocation to Subscriber 1



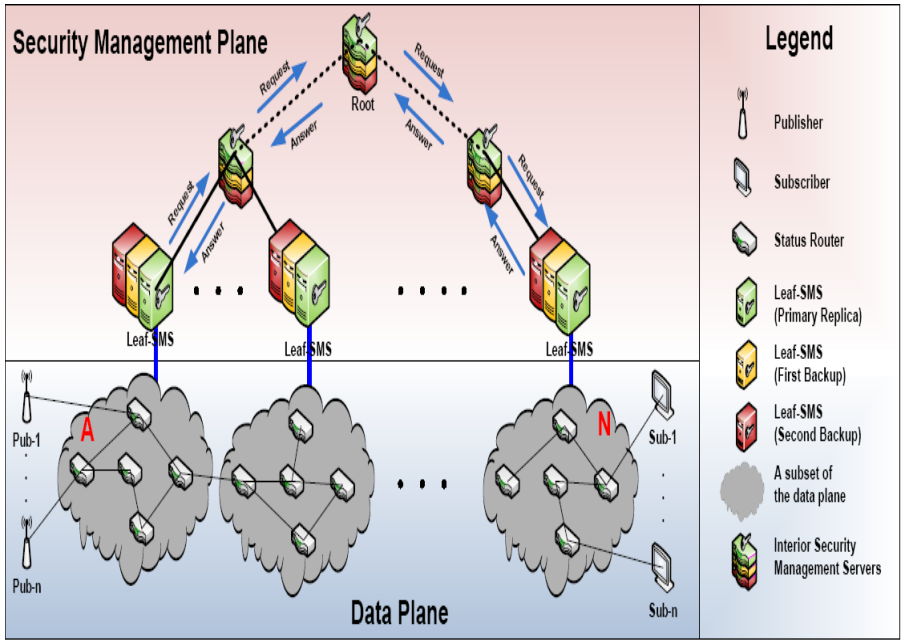
Route Allocation to Subscriber 2



Note: Sub2 may have a different rate, latency, or redundancy than Sub1

Security Management System

- Defend against attacks on confidentiality, integrity, accessibility
- Evolve with changes in the security field (securely upgradeable modules)
- Constant cost end-to-end regardless of number of hops
- Support multicast, redundant paths and call backs
- Protect private utilities' business sensitive information



Ongoing Security Research

- Authentication is the first step towards meaningful, secure communications
- Any amount of encryption is not useful on its own unless participants are authenticated
- Preserve the evolvable nature of the security system, extend it to authentication
- Sustain key material for a longer period of time without loss of efficiency or protection
- Maintain hierarchy of control, support private utilities' business sensitive information protection

Multi-Level Contingency Plannning & Adapting

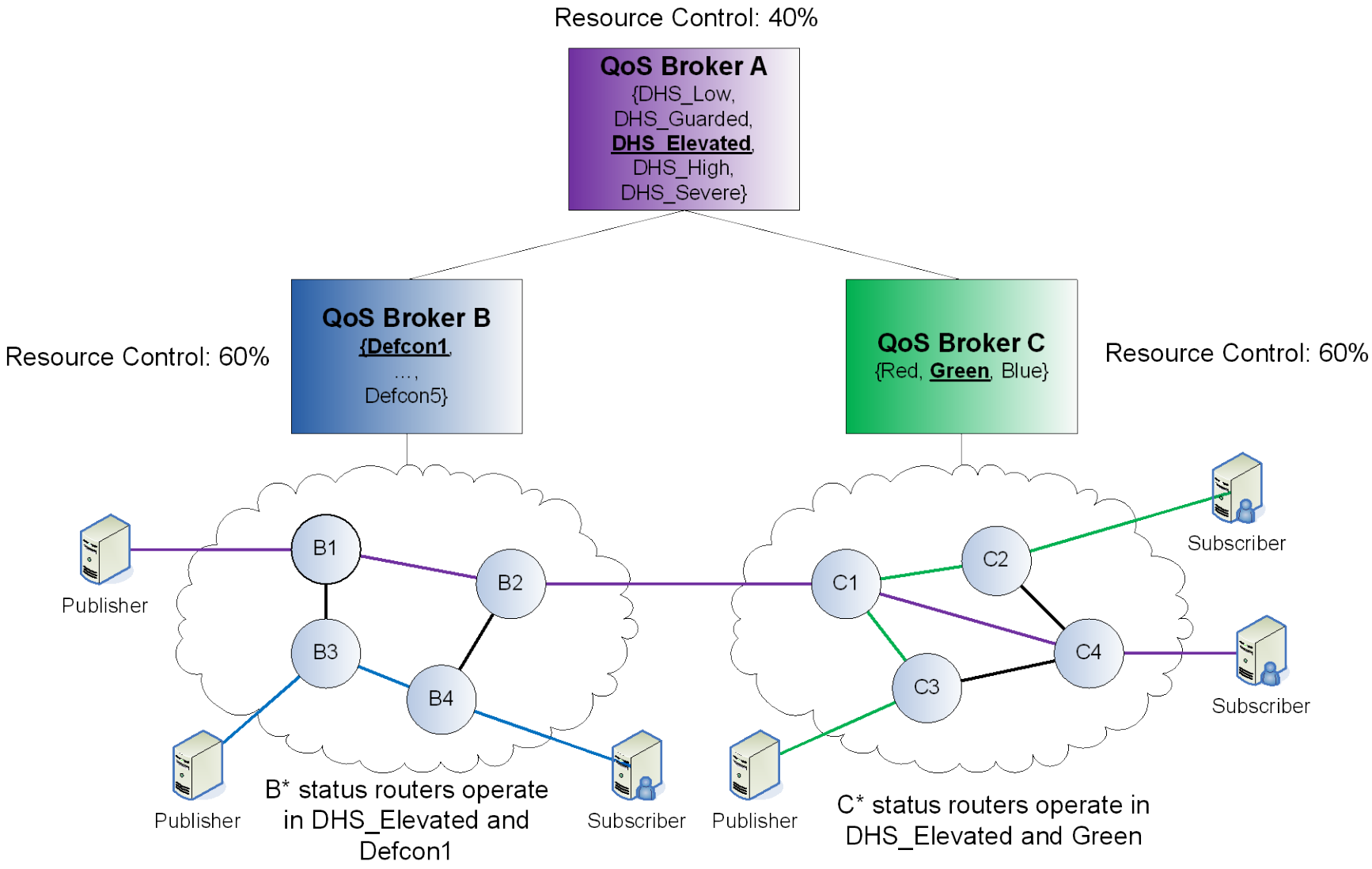
- GridStat supports **operational modes**
  - Can switch routing tables very fast
  - Avoids overloading subscription service in a crisis
- Electricity example: Applied R&D on coordinated
  1. Power dynamics contingency planning
  2. Switching modes to get new data for contingency
  3. New PowerWorld visualization specific for the contingency

  involving contingencies with
  A. Power anomalies
  B. IT failures
  C. Cyber-attacks
- Note: state of art and practice today: 1 & A only, offline

Mode Terminology

- A mode is a set of routing tables
  - Contains forwarding rules for a bundle of subscriptions
- Every QoS broker has it own *mode set* and will always operate in one of those modes
- Every status router:
  - Uses as many routing tables for forwarding status events as there are *levels* in the management hierarchy
  - Has all routing tables defined in its ancestor scope pre-loaded in memory (or on disk)

Mode Terminology



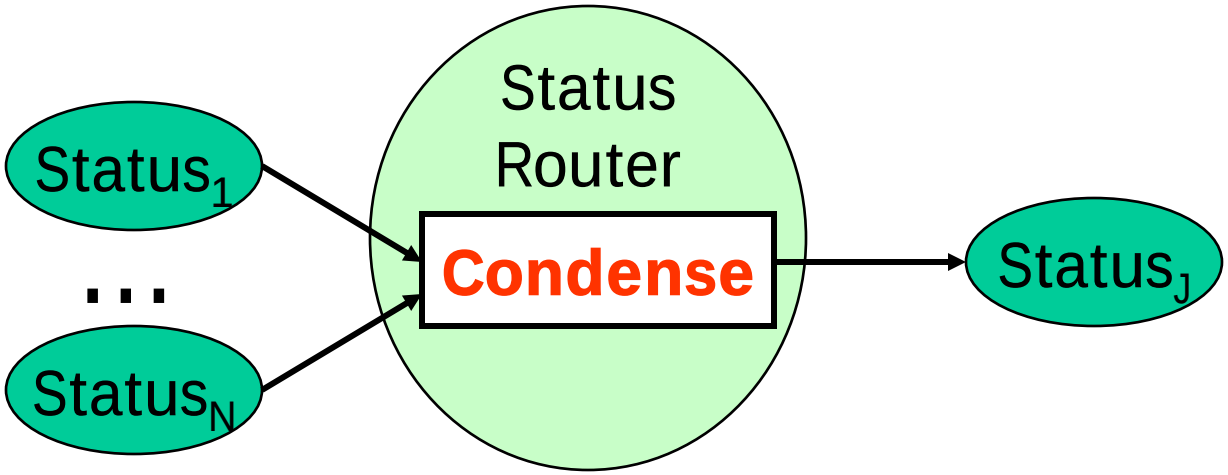
Mode Change Algorithms

- Mode change algorithm: coordinator contacts a set of status routers and informs them to switch routing tables
- Mode change algorithms with different tradeoffs
  - Hierarchical mode change algorithm
    - A -> B: Enables subscriptions in modes A and B to flow
    - Correctness vs. performance
  - Flooding mode change algorithm
    - Best-effort algorithm: status router network switches modes at a future timestamp
    - Benefits from the amount of redundancy the status router network provides
    - Optimal delay (shortest path)

Data Load Shedding

- Electric Utilities can do **load shedding** (I call **power load shedding**) in a crisis (but can really hurt/annoy customers)
- GridStat enables **Data Load Shedding**
  - Subscriber's desired & worst-acceptable QoS (rate, latency, redundancy) are already captured; can easily extend to add priorities
  - In a crisis, can shed data load: move most subscribers from their desired QoS to worst case they can tolerate (based on priority, and eventually maybe also the kind of disturbance)
  - Works very well using GridStat's operational modes
  - Note: this can prevent **data blackouts**, and also does not irritate subscribers
- Example research needed: systematic study of *data load shedding* possibilities in order to prevent *data blackouts* in contingencies and disturbances, including what priorities different power apps can/should have…
- Lets critical infrastructures adapt the data communiations infrastructure to benign IT failures, cyberattacks, power anomalies, …

Condensation Functions



- *Condensation functions* allow applications to define new derived status variables
  - Sometimes subscribers just read a large set of status items once to calculate a derived variable
  - Supported by allowing user-defined condensation functions to be loaded in status routers
  - Building block for other mechanisms/capabilities
- Can be dynamically loaded into SRs (or elsewhere)

Overview of Other Mechanisms & Features

- Subscriber-side caching
  - Can get callbacks, instead (database integration)
- Subscriber-side cache extrapolation
  - Predefined primitives
  - User-defined object
- Alerts
  - Subscribed (like boolean status variable)
  - Flooded
- Actuator RPC with safety
  - Client-server request-reply delivered over multiple, one-way, GridStat update paths
  - Pre-conditions: abort call if sanity check fails
  - Post-conditions: additional physical verification call succeeded