S-38.3157

Protocol Design

2005–2006, 4th period

Jörg Ott jo@netlab.hut.fi SE 324
Carsten Bormann cabo@tzi.org

General

- Architectures, mechanisms, principles, issues, and pitfalls for protocol design from a conceptual viewpoint (examples!) (taking an Internet perspective)
- Lectures: Tuesday, 14 – 16, S2 and Thursday, 12 – 14, S2
- Exercise (assignments + practical stuff): Thursday, 14 – 16, E111
- Prerequisites
  - S-38.(2)188 (or equivalent knowledge)
  - Interest in protocols and their technical realization
  - Substantial coding skills (no novice in C/C++, Java, … for communications)
- Suitable for graduate and postgraduate studies: 4 ECTS points
- 8 weeks of an experiment (new for all of us)

Theoretical and Practical Assignments

- 3 (or 4) Assignments
- Practical Assignments with theoretical documentation / motivation
  - The practical coding assignments building on top of one another
  - Create the structure of a communication application
  - Deal with socket i/o and related system calls
  - Support parameterization and some visualization (no GUIs!)
  - Make design choices for a small protocol (and possibly regret them later)
  - Document (motivate and defend) parts of your design in writing
- C/C++, Java, Perl, Ruby, … (choose your favorite language) code
  - Write portable applications to be run on several machines accessible from the university
- Small groups: 3
  - Send one email per group in the following format (one line per group member)
    "Last name: First name:IDs:email address"
- Completion: usually 2 weeks, last one until 5 May 2006
  - Send email with tgz or zip archive of source, build environment
  - Result review yet to be decided (possibly at last Thursday in auditorium)

Theoretical Assignment

- Closer to the end of the course
- Analyze an IP-based protocol with respect to the protocol design aspects we will have discussed
  - Keep in mind the Internet architecture and design principles
- Grading of assignments based upon theoretical parts
  - Will add points to the final exam (about one third)
- Practical assignments must be completed
  - They should be easier to deal with than those in NMPS (S-38.3150)
- 50% of the points from the theoretical assignment parts required

Exam

- 11 May 2006, 13 – 16, S3
- 3 hours time
- Hints in the last lecture
- Grade based upon the exam plus theoretical exercises
  - Delivering working practical assignment results prerequisite

Material

- Slides will be online as PDF
- Primary literature: RFCs, Internet Drafts, research papers
  - We will point to some recommended ones for studying
  - Do-it-yourself: google, ACM & IEEE digital library, …
- Books
  - There are some old ones (beginning to middle of the 1990s)
    - Different focus than the course: mostly on mechanics and approaches
    - Not so much about design principles and experience
  - Sometimes individual chapters in books have useful contents
    - Example: Radia Perlman: Interconnections: Bridges, Routers, Switches, and Internetworking Protocols, 2nd Edition, 1999. Chapter 18 (available online)

Relation to other Netlab Courses

- 38.(3)188: Computer Networking: prerequisite
  - Some minor overlap
- 38.(3)115: Signaling Protocols: complementary
- 38.3150: Networked Multimedia Protocols and Services: complementary
  - Can be done before or afterwards
  - Helpful if done before
- S-38.3155: Seminar on Challenged Networks
  - Postgraduate seminar, next term, 1st period
  - Looks at particular environments for different style of protocol design
- S-38.(3)158: Protocol Design – Practical Assignment: will likely not happen

Contents 1

1. State sharing and reliability
2. Scalability concerning many dimensions
3. Resource consumption and fairness (network and endpoints)
4. Protocol syntax and encoding
5. Security 1: Robustness
6. Interoperability, Evolveability
7. Security 2: Cryptographic techniques

Contents 2

Further Information

- Course web page
  - http://www.netlab.hut.fi/~jo/teaching/pd/index.html
- Newsgroup
  - opinnot.sahko.s-38.tietoverkkotekniikka
- Material and other resources will be placed on the course page

And we need your feedback!

Protocol Design

Overview and Course Focus

Motivation: Why still Protocol Design?

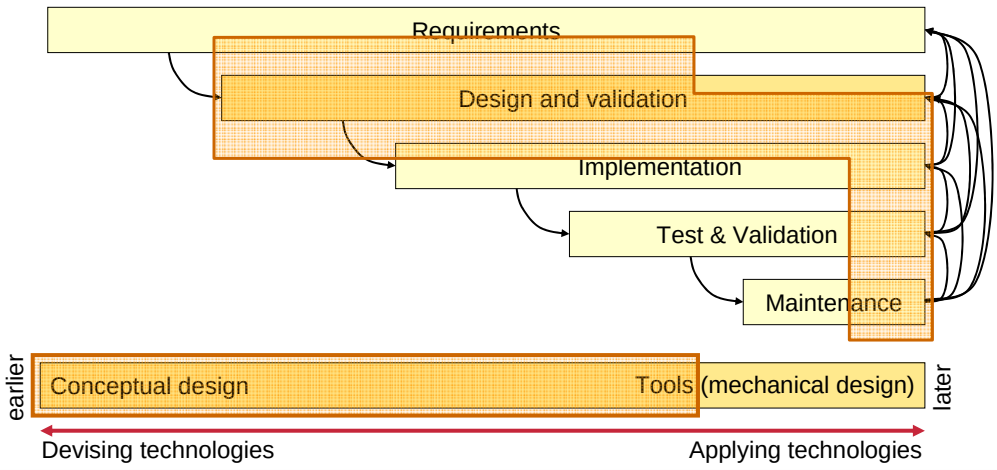
- New applications appear all the time – more and more net-based
- Within applications, functional decomposition and distribution makes protocol design an inherent part of system design
- Evolution of communication technology incurs new demands
- Environmental changes require reconsidering the design of existing protocols
- Migration (aka “convergence”) requires re-thinking solutions to old problems for a new environment (e.g. IP telephony, IP TV)
- Vast variety of problems and solutions
  - Simple (e.g., just use RPC) vs. complex (BGP-4 for telephone numbers)
  - All layers (from wireless MAC to QoS to autoconfiguration to applications)
  - Closed environments (within a product) to open standards

What is Protocol Design?

- Many possible views
  - Mathematical modeling
    - Design and correctness proofs
  - Protocol engineering process
    - Management and process aspects of protocol design (software engineering view)
  - Building blocks and design patterns
    - Mechanisms for certain functions in creating protocols
  - Tool chains for protocol specification, implementation, and validation
    - Automating the creation process (but not the conceptual thinking)
  - …
- We are interested in
  - Why some designs work better (get accepted) than others (which don't)
  - Ideas of what is known as good practice beyond the engineering literature
  - Understanding relationship between functional and non-functional aspects
  - Considering some non-technical real-world aspects as well

Sample Protocol Design Process

(just a random diagram – variation of the waterfall model)



Requirements Aspects

- Understanding which problem to solve
  - Real problems vs. thoughts about solutions in search for a problem
- Understanding the requirements
  - Functional: features, security, …
  - Non-functional: scale, operational aspects, time-to-market, cost
- Understanding the constraints
  - Functional: operational environment
  - Non-functional: cost, weight, energy consumption, memory, CPU, …
- Understanding the acceptable tradeoffs
  - Must vs. nice-to-have
- Is this some special case of a more general problem?
  - If so: does the problem become *simpler* by generalizing?
    If not, is the more general problem worth solving?

Some General Protocol Design Aspects (1)

- Design scope
  - Part of a specific application design
  - Creation of a platform for a competitive environment
- Design target
  - Complete solution, e.g., for an application
  - Creation of building blocks targeted at flexible re-use
  - Use of building blocks or technologies to create a particular solution
- Important design decision: Make or take
  - Re-use existing technologies (accept less than 100% match)
    - Benefit from experience, code, etc.
    - But: who has change control, how long will the technology be supported, does it *really* fit, will both protocols evolve in parallel, …?
  - Create new technology from scratch (accept higher risk, longer time to market)

Some General Design Aspects (2)

- Learning from solutions to related problems
  - Borrow concepts and mechanisms – but only where applicable!
  - Avoid mistakes. Look at real-world deployments before borrowing
  - Yet avoid the "second system syndrome"
- Remember requirements during the design phase
- Some simplified meta rules ("protocol folklore")
  - Optimize for the common case (if at all)
  - Don't overengineer – Keep it simple stupid (KISS)
  - Avoid options and parameters
  - Remember that it needs to be implemented in the end

(we will address these and more such issues during the course)

Some General Protocol Design Aspects (3)

- Separation of concerns
  - Treat and solve independent aspects independently
  - Caveat: what is *really* independent?
- (Strict) layering
  - Block box, well-defined service access points (SAPs) with layer-internal protocols
  - Intends to completely shield lower layers and communication details from higher layers
- Leaky abstraction
  - Strict layering will not always work, particularly if things go wrong
  - Expose issues rather than trying to conceal them at any cost
  - Applies to protocol design, to coding (and code generation), and others
- Cross-layer optimization gaining importance
  - Deal with dependencies on the lower layers
  - Limit: your system is not always directly connected to the weakest link (layer)

Design Validation

- Protocol design is relevant to later protocol validation
  - From a correctness perspective
  - From a performance perspective
- Correctness of a specification
  - May involve formal specification as design methods
    - Using your favorite modeling or specification language
  - May involve formal proofs
    - Mostly for “simple” protocols and problems
- Performance of a specification
  - Mathematical modeling and analysis
  - Evaluation by means of “implementation” and simulation
- Both validations provide important feedback for the design process

Implementation & Validation

- Protocol implementations need to be correct and interoperable
  - Beware of complexity!
  - In some cases, code may be generated from specifications using tools
- Again: validation
  - Limited functional validation through testing
    - Test cases may be generated from specifications
    - Usually cover only usage scenarios of limited complexity (explosion of number of tests)
  - Performance validation through emulation and field tests with measurements
- Difficulty: getting even close to the real-world conditions (in the lab)
  - True validation will only occur through real world deployment (“in the wild”)
  - Different platforms, different implementations, different user behavior, different environmental conditions, (different interpretations of the spec), …
    - Will also tell something about the impact on the network at large
- Implementation experience provides most important feedback

Conformance vs. Interoperability

- Traditional thinking:
  - All implementations must conform to specification
  - If specification is *good*, this ensures interoperability
- Modern thinking:
  - Implementations have errors
  - Specifications have errors and ambiguities
  - Interoperability is actually more important than conformance
    - This includes interoperability with erroneous, but deployed systems

Operations and Maintenance

- Rollout
- Monitoring
  - Protocol and device operation
  - Its impact on its environment
- Diagnosis, Debugging
- Protocol evolution over time
  - To fix bugs
  - To meeting changing or new requirements
    - To get rid of unnecessary requirements and constraints
  - To deal with changing environmental conditions

A Note on Protocols in the Real World

- Protocol design usually makes assumptions
  - About the environment it will operate in
    - Technical terms: packet network, delay, packet loss, MTU, range of data rate, etc.
    - Organization terms: trust, common management, configuration, interaction, etc.
  - Lower layer services and characteristics to build upon
  - Higher layer applications using it
- Protocols may be successful or even “hyped”
  - Examples today: HTTP, SIP, XML, to some extent SOAP, …
- If they are, they will be used outside their specified limits
  - In different environments, at different scales, for different purposes, …
- People will blame the designer if they don’t work properly then
  - Applicability statements are not necessarily read or adhered to

Some Examples for who does Protocol Design

- A (formal) standards body
  - Without link to reality: driven by formal processes and voting
  - With link to reality: driven by perceived needs, well-defined deliverables
  - Worry about network and protocol architecture at large
- An industry consortium to make the market grow
  - Driven by (artificial, perceived) deadlines and limited by compromise
  - Worry about system architecture in a given market segment (to suit their needs)
- A group in an enterprise trying to get a specific problem solved
  - Driven by immediate (and mid-term) customer needs
  - Worry about product architecture and environmental constraints
- Researchers/scientists
  - Driven by solving complex problems in an elegant way
    - May be tempted to get 110% of a solution for some problem aspect (not necessarily for all)
  - Biggest potential for long-term architectural thinking (often not considered)

Subject Areas of Protocol Design

- General design space
- Functional building blocks
- Meta design aspects

Protocol Design is about Trade-Offs…

…given sets of requirements and environmental constraints.

- "Good, fast, cheap – pick two, you cannot have all three."
- Examples
  - Reliability vs. delay
  - Functionality vs. bandwidth
  - Extensibility vs. efficiency
  - Functionality vs. simplicity
- Virtually any design decision taken to achieve one goal will counteract another
  - Need to find a reasonable compromise to achieve desired function at acceptable cost

Where Theory meets Practice…

- Many design rules for protocols can be found
  - Mechanisms to achieve certain functionality
  - Keep it flexible and extensible
  - Make it effective and efficient (optimize)
  - Make it resilient
  - …
- To be applied wisely (not blindly)
  - Considering the trade-offs
- Beware of complexity
  - People will blame the their device or technology if the stuff doesn't (inter)work
    - Regardless of where the problem is
  - Too expensive or too difficult to use
- Premature optimization is the root of all evil
- …

Communication Partners and their Roles

- Point-to-point vs. multipoint communications
  - How many parties are involved in the protocol (from a semantics perspective)?
- Unicasting vs. group-overlays vs. multicasting
  - What type of information exchange is assumed?
- Client-server vs. peer-to-peer communications
  - Are the involved parties "equal" or do they have different responsibilities
    - Note: peer-to-peer is more general than today's widespread applications)
  - In case of groups: are some more important than others?
    - More than just two different classes of peers
- End-to-end vs. intermediaries vs. router-assist
  - What kind of entities may, are, or must be involved? Are the "visible" or not?
- Communication among end systems vs. among network elements
  - Transport and application vs. routing, network, maintenance protocols
- End-to-middle communications

Identifying Communication Partners

- Names
  - Human readable identifiers that can be remembered! (e.g., DNS name, URI, URN)
- Identifiers
  - Machine-processable identifier (e.g., HI)
- Addresses
  - Protocol-level identifier (e.g., IP address)
- Locators
  - Information about the location of a partner in the network topology
- Need to be managed (unique assignment)
  - Or chosen randomly (and defended) in ad-hoc environments (birthday paradox)
- One needs to resolved into the other
  - Address books, (distributed) data bases (e.g., DNS, DHTs), protocol exchanges, caching, (manual) configuration, …

Some Protocol Design "Knobs & Variables"

- Stateful vs. stateless operation
  - How much information is preserved across information exchanges
    - Notion of an "association" or a "connection"
  - Where is this state kept (one or both peers in the point-to-point case)?
- Fixed nodes vs. mobile nodes
  - impact on routing, reachability, …
- Wireline vs. wireless communications
  - Implications of different link layer technologies in general
- Infrastructure-based vs. ad-hoc/autonomous communications
  - What types of infrastructure are assumed? (e.g., routing, naming)
- Security within the protocol vs. relying on security elsewhere
  - Which implications (e.g., for required infrastructure such as PKI)

Functional Building Blocks (1)

- Naming and addressing
- Rendezvous or invocation mechanisms
- Semantics and properties of protocol operations
  - Idempotent operations, delta vs. full state updates, synchronization, …
- Interaction paradigms
  - Synchronous, asynchronous, both
  - RPC-style operation vs. event notifications at any time
- “Reliability”
  - Includes flow control, sequence preservation, etc.
  - How probable is it that a certain operation will not fail.
  - Example: transferring a file from A to B unchanged
  - Multitude of different protocol mechanisms available

Functional Building Blocks (2)

- Multiplexing
  - Within the application protocol vs. using lower/requiring higher layer mechanisms
- “Multi-threading”
  - Allowing multiple ongoing interactions at the same time
  - E.g. lock-step vs. “windowing”
- Security
  - Authentication, integrity, non-repudiation (sender, receiver), confidentiality
  - Authorization of operations
- (Auto)configuration
  - How to get a system into a working condition

Meta Aspects of Protocol Design (1)

Independent of specific functions, yet to be provided in line with the respective protocol

- Adaptivity
  - Capability of adapting to different environmental conditions (typically “QoS”) (graceful degradation of service as long as acceptable)
    - Example: playout delay and codec adaptation with IP multimedia
- Scalability
  - Capability of working across a wide range of environmental parameters
    - Typical example: Number of operational nodes
    - Data rate, error rate, path length, delay (see above)
    - Number and size of data items
- Efficiency
  - Maintaining a reasonable level of overhead
    - Example: protocol encoding, protocol headers

Meta Aspects of Protocol Design (2)

- Performance
  - Number of protocol interactions, packets, bits, processing
  - But don’t optimize (too early in the process)!
- Security (again!)
- Deployability
  - One special case: robustness (against DoS, single point of failure, etc.)
  - Another special case: ability for stepwise introduction into the real world
- Evolvability
  - Backward and forward compatibility
- Operability and manageability