

---

# DTN Reference Implementation

Michael Demmer

DTNRG Meeting – San Diego, CA  
August 6, 2004

# Motivations / Background

---

- Difficulties with current implementation:
  - Few internal interfaces and abstractions, hence hard to modularize or extend
  - Written in C, thus little language support to help this problem
  - Much of the code is related to management of custom, hand-tooled data structures
  - In parts, the code has been tuned for memory footprint and forwarding performance, at the expense of clarity
  - To meet specific demo requirements, some features added outside the scope of the architecture and bundling specs

# Motivations / Background (2)

---

- Decision: rewrite the reference implementation
  - Keep focus on goal of a *reference implementation*, prioritize features and design decisions accordingly
- But beware of reinventing the wheel
  - Use experiences (and code) from current version where applicable
  - Avoid second system effect and feature creep

# High Level Design Goals (in order)

---

1. Code Clarity
2. Subsystem Modularity
3. Flexibility in deployment
4. Ease of extension and prototyping
5. Scalability / Performance / Code Footprint

# High Level Design Decisions

---

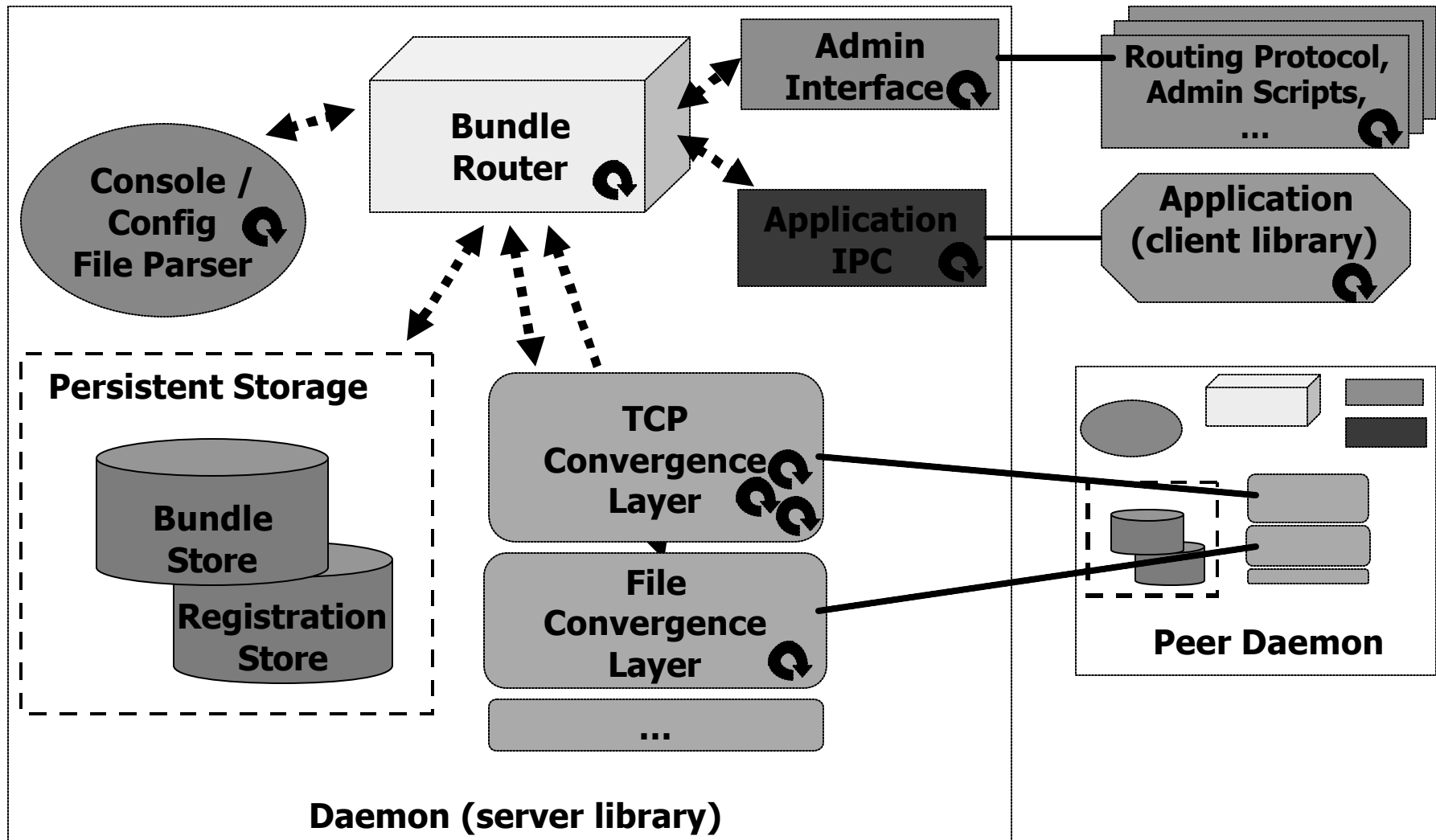
- Rewrite in C++, use STL data structures
  - Common data structures and idioms improve readability, conciseness, and makes more of the code related to bundling
  - Language features vastly simplify interface virtualization
- Diligent, clear module boundaries
  - Lots of internal interfaces between components
- Multiple persistent storage implementations
  - Berkeley DB, Postgresql, Mysql, UNIX File
  - Eases requirements for adaptation to different environments

# High Level Design Decisions (2)

---

- All Bundle metadata is always in memory
  - Copy of all “live” bundles stored in the persistent database
  - Simplifies memory management (at the expense of scalability)
  - Payloads may be in memory or on disk, depending on size
  - Bundles are reference counted, keep backpointers to containers
  - On initialization, all bundles re-read from the database and essentially re-routed to their appropriate contact lists
- Forwarding decisions (policy) separate from actual forwarding implementation
  - Routing module interacts through a clearly defined event dispatch interface

# Bundle Daemon Components



# Specific Components – Routing

---

- BundleRouter makes *all* decisions related to bundles
- Dispatch interface to receive internal events:
  - bundle arrival / transmission / expiration
  - application registration / cancellation
  - contact open /closed / interrupted
  - custodian acknowledgement, expiration timer, etc...
- From event input, router builds action list
  - Action list is passed to a BundleForwarder module that implements the decisions outlined in the instructions
  - Alternatively, embed router module in a simulation framework without changing the router code
  - Potentially run multiple routers in parallel, with one active and a set of passive ones to be compared side-by-side

# Specific Components – Routing (2)

---

- Challenge: action list specification that can handle a wide range of instructions:
  - E.g. send bundle to one of N contacts, maybe the first available, maybe the one with the most capacity
  - E.g. queue bundle for a contact, but only activate the channel if a threshold limit of queued data is reached
  - E.g. a registration that gets to peek at all bundles that flow by
- Claim: enforcing this policy / action separation results in clearer, more flexible implementation
  - Forcing function to enumerate the potential routing decisions
  - Allows forwarding to evolve independently from routing

# Specific Components – Persistent Store

---

- Abstraction layer virtualizes different underlying technologies
  - Berkeley DB, Postgresql, Mysql, UNIX Files
  - Clear interfaces allow new technologies to be easily added
- Semantic interfaces for each storable element
  - `BundleStore`, `RegistrationStore`, `GlobalStore`, ...
  - Semantic functions allow leveraging of underlying capabilities, e.g. `delete_expired_bundles` can use a SQL query
    - “`DELETE from bundles where expired < now`”

# Specific Components – Persistent Store (2)

---

- Each storable element implements a single function to store, update, and retrieve records for all underlying implementations:

```
void
Bundle::serialize(SerializeAction* a)
{
    a->process("bundleid",      &bundleid_);
    a->process("source",       &source_);
    a->process("dest",         &dest_);
    a->process("custodian",    &custodian_);
    a->process("priority",     &priority_);
    a->process("custreq",      &custreq_);
    a->process("custody_rcpt", &custody_rcpt_);
    a->process("return_rcpt",  &return_rcpt_);
    // ...
}
```

# Specific Components – Client API

---

- DTN applications link with a thin library of API functions for bundling
  - API implements a simple IPC protocol
  - Uses XDR routines from Sun RPC for argument marshalling (but not the RPC protocol itself)
  - Also supports direct linking of an application with the daemon library, for a single app per host deployment model
- API modeled after sockets
  - `dtm_socket`, `dtm_bind`, `dtm_send`, `dtm_recv`, `dtm_poll`,...
  - Leverage programmer experience and familiarity with API
  - Still have DTN specific functions like registration

# Current Status

---

- Completed work (after ~1.5 months)
  - Bundle wire Protocol v3 (most of it)
  - SQL persistent store
  - Static routing module
  - Simple TCP convergence layer
  - Internal interfaces for Bundles, Contacts, Registrations, Convergence Layers, Routing, Events, etc
  - Command line interface and configuration
  - Utility and debugging classes “borrowed” from other projects

# Current Status (2)

---

- Big ToDo list
  - Client API and IPC protocol
  - More work on TCP Convergence Layer to deal with interruptions, implement a real framing protocol, acks, etc
  - Other convergence layers: UDP and files (sneakernet)
  - Fragmentation interfaces and implementation
  - Other storage implementations (Berkeley DB, UNIX files)
  - More advanced routing algorithms
  - Expiration
  - Custody transfer
  - Delivery status notification
  - Security / Authentication
  - Everything else not in this list...

# Backup

---

# Specific Components – Admin Interface

---

- Embedded Tcl interpreter for config file and command line interface
  - Easy to add and extend commands, both for configuration and for debugging, profiling, etc
  - Can support telnet to an admin port for remote console
- Optionally support external routing protocols
  - Simple interface to manipulate the routing tables, routing protocols and logic can be implemented in a separate process
  - Similar to UNIX `gated` interacting with kernel routing table