Building a Reliable Multicast Service
based on
Composite Protocols

Sandeep Subramaniam

Master's Thesis Defense

The University of Kansas

06.13.2003

Committee:

Dr. Gary J. Minden (Chair)

Dr. Joseph B. Evans

Dr. Perry Alexander

Outline

- Motivation
- Composite Protocol (CP) Framework
- Design of a composable service – multicast example
- Implementation of service over Ensemble
- Testing and Performance Evaluation
- Summary & Future Work

Basic definitions

- Protocol component
  - Single function entity that embeds minimal protocol functionality
  - E.g. checksum, reliable-delivery, fragment
- Composite protocol
  - Collection of **protocol components** arranged in orderly fashion
  - E.g. An IP-like composite protocol would consist of forwarding, fragment and checksum protocol components.
- Composable service
  - Collection of 2 or more co-operating **composite-protocols**.
  - E.g. multicast consists of multicast routing, group management and replication of data

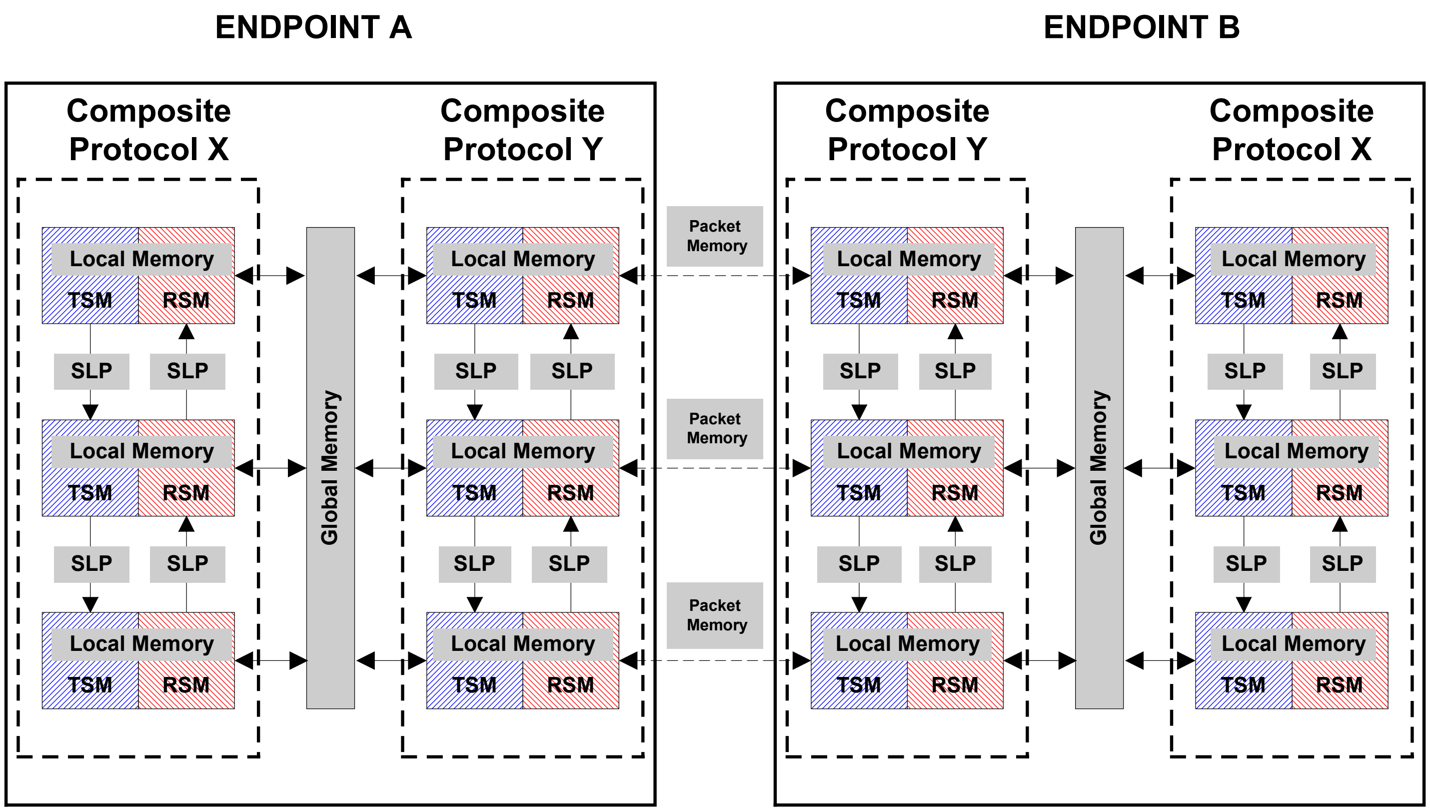
Advantages of Composite Protocols

- Protocols implemented as collections of single-function components
  - Reusability
  - Flexibility
  - Aid in formal verification, building correct protocols.
  - Customization , fine tuned protocol stacks
  - “Properties-in Protocol-out”

Motivation

- Motivation for composable service
  - Apply the composite protocol approach to wider range of protocols
    - Data and control plane protocols
    - demonstrate feasibility and applicability
    - Expand the library of protocol components
  - Address issues of inter-protocol communication
- Building a network service addresses all above needs.
- Reliable multicast service chosen as an ideal example
  - 3 co-operating protocols operating in tandem
    - Reliable replication of data (multicast forwarding)
    - Multicast routing
    - Group management

Composite Protocol Framework



Design of a Composable Service - Steps

- Decomposition
  - Identify key functional entities in the monolithic counterpart.
- Specification of protocol components using AFSMs
  - Represent each component as a pair of SMs (TSM & RSM)
  - Specify local, SLP, packet and global memory requirement
  - Identify data and control events
- Building the stacks
  - Linear composition to yield composite protocol
- Deployment
  - stacks to place on a particular network node
- Global memory objects for inter-stack communication

Step 1 - Decomposition (Multicast Service)

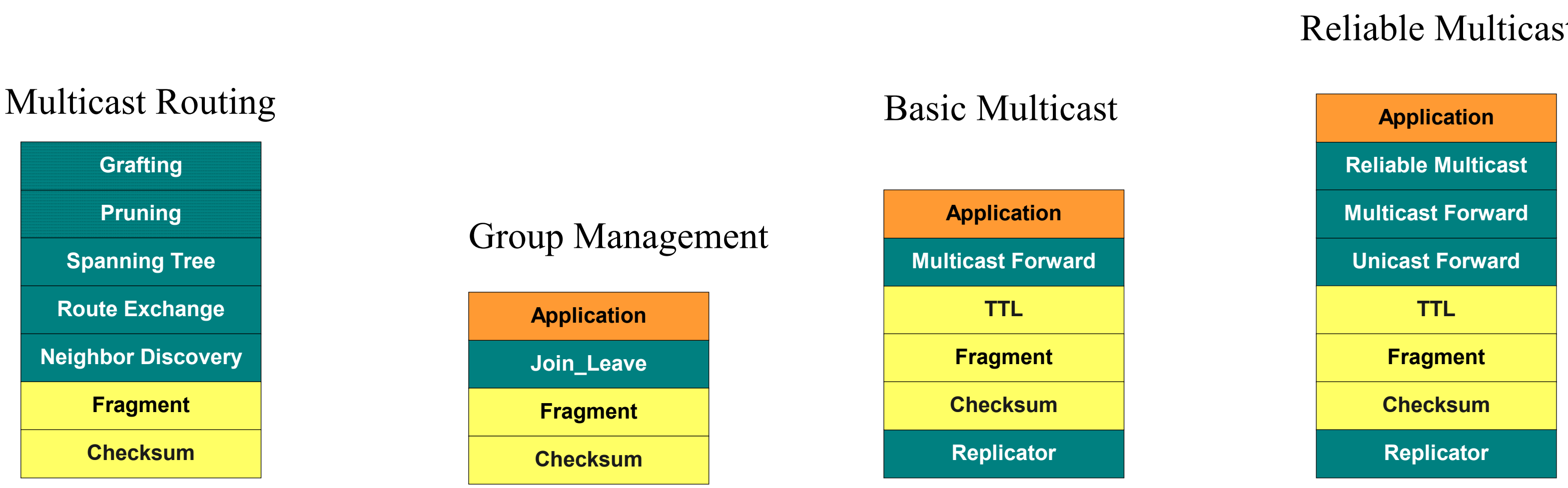
- Multicast routing based on DVMRP
- Group Management based on IGMP
- DVMRP
  - Neighbor Discovery
  - Route Exchange
  - Spanning Tree
  - Pruning
  - Grafting
- IGMP
  - Join/Leave
- Data
  - Multicast Forwarding
  - Reliable Multicast
  - In-order Multicast

Step 2 – Specification

- Each functional component has to confirm to CP specs.
  - Independent of other components
- Each protocol component specification consists of
  - Pair of AFSMs – TSM and RSM
  - Memory requirements
    - Packet memory – bits on the wire or component header
    - Local memory – maintaining local state information
    - SLP Memory - memory local to the stack but pertaining to packet
    - Global memory – external memory requirements
  - Events:
    - Data: packet arrival from component above/below
    - Control : timers , application-component interaction
- Specify assumptions and parameters

Building the Stacks – Step 3

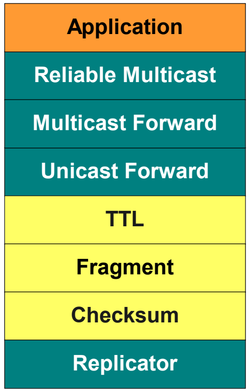
- Group related components into composite protocol - linear composition
- Try re-using existing components



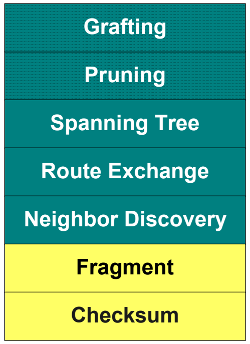
Building the Stacks - Stack Ordering

- Determine the order of stacking among components
- Does order matter ?
- Property – Oriented
  - Layer N provides a property to Layer N+1
  - Order of component determines stack behavior
  - E.g. reliable multicast stack
- Control – Oriented
  - Components in stack are independent
  - Layer N does not provide specific property to Layer N+1
  - Order may affect performance not stack behavior
  - E.g. multicast routing stack

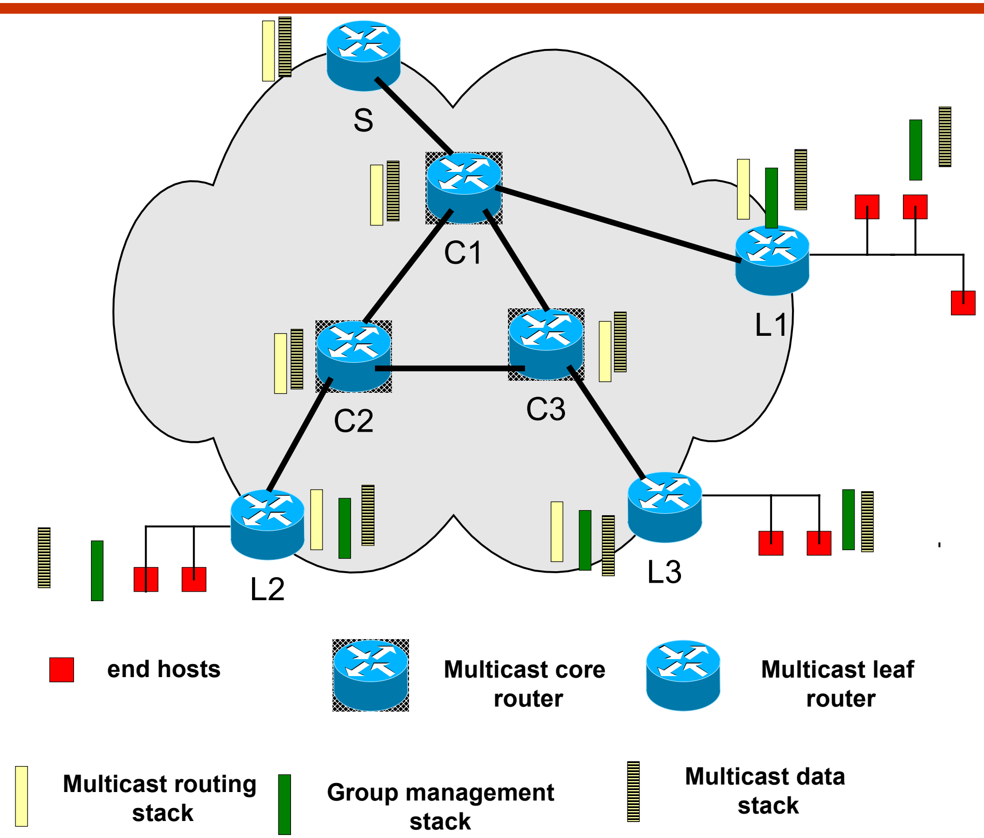
Reliable Multicast



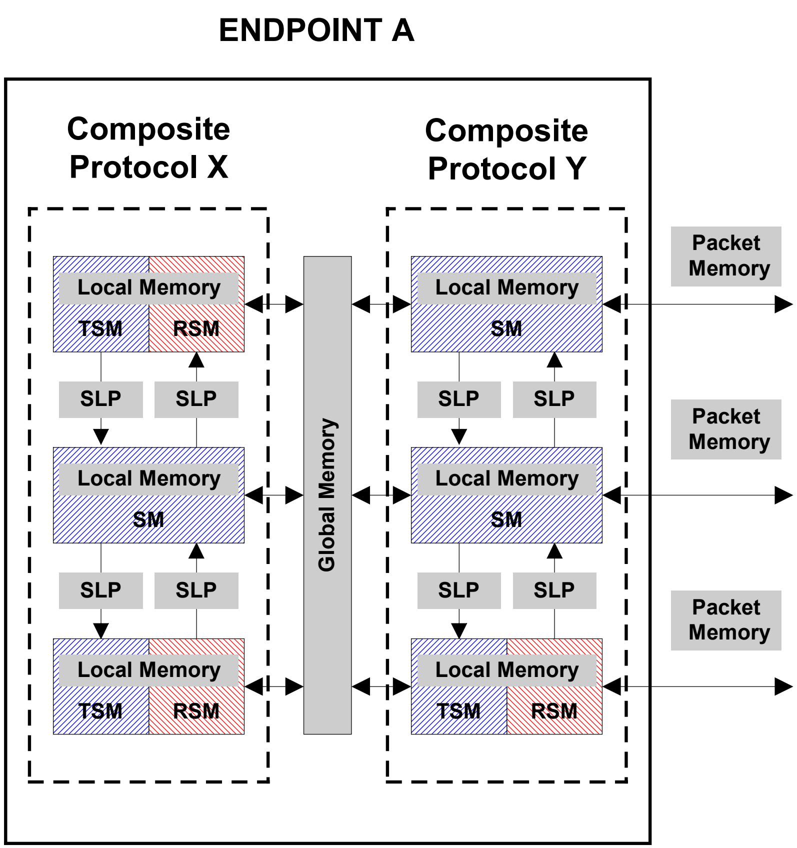
Multicast Routing



Deployment – Step 4



Inter-stack Communication - Global Memory

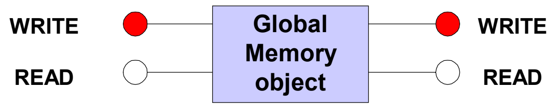


- Addresses the issue of cross-protocol communication
- Acts as a repository for data shared among different stacks.
- Accessible to all components of all composite protocol instances at that endpoint
- Scope and extent greater than any single protocol accessing it.
- Functional interface for read/write
- Responsible for initialization and maintenance of shared info

Global Memory – Features

- Separation of Protocols and Data Management
  - Independence between protocols and global memory data management
  - Protocol component expresses requirements for global memory access through its external functions
  - Protocols that write to /read from global objects need not agree on internal data format

- Functional Interface
  - Access to shared data only through write/read functional interfaces
  - Encapsulates shared data
  - Hides internal representation of global memory object



Global Memory - Features

- Synchronization
  - Each object solely responsible for providing synchronized access to its shared data
  - Synchronization not delegated to users of the shared object.
  - Access control mechanism is implementation specific
  - Semaphores or other mechanisms can be used

- Extensibility
  - Object definition can be extended
  - Internal data structures / external functions can be added
  - Backward compatibility easily maintained

Implementation

- Overview of Ensemble
- Global Memory Implementation
- Operational overview of service
- Protocol Interaction through global memory

Ensemble

- Group communication system developed at Cornell
- Used as base framework for building composite protocol
- Reasons:
  - Written in OCaml functional programming language, aiding for formal analysis of code
  - Ensemble uses linear stacking of layers to form stack
  - Event handlers executed atomically
  - Unbounded message queues between layers
  - Provides uniform interface
  - Support for dynamic linking of components , adding/removing components from stack at run-time

Multicast Global Memory Objects

- Neighbor Table
  - Stores 1-1 mapping between an interface and neighbor discovered on that interface
- Routing Table
  - Repository for unicast routes
  - Metric and next-hop information for each route prefix stored
- Source Tree
  - Maintains spanning tree for each multicast source in the n/w
  - Contains list of dependent downstream neighbors for each source

Multicast Global Memory Objects (contd)

- Prune Table
  - Contains core and leaf interface prune-state information for each (source/group) pair in the n/w
  - Interfaces can be in 3 states : un-pruned/pruned/grafted
- Group Table
  - Stores current list of group members on each leaf interface
  - Updated when members join/leave groups

Global Memory – Linux Shared Memory

- Shared memory – fastest form of IPC
  - Single chunk of memory shared by 2 or more processes

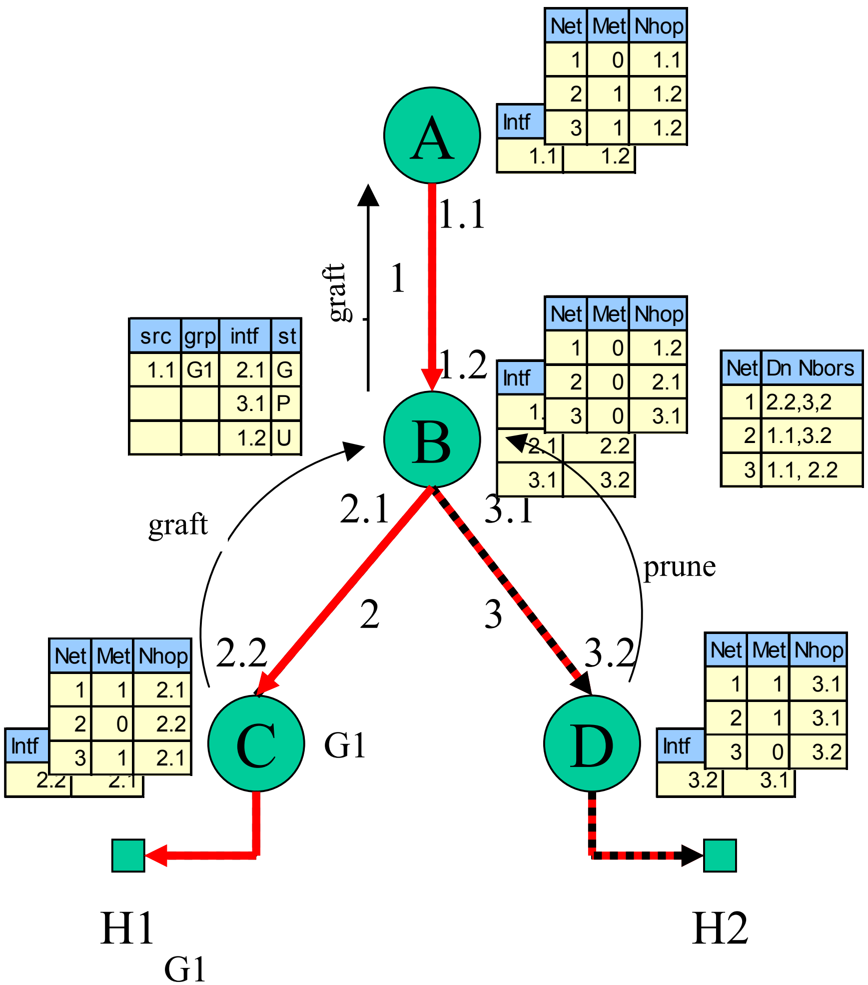
- Steps in creating a global memory object
  - Specify read/write functional interface using CamlIDL
  - Implement functions using Linux shared memory system calls
  - Handle concurrency issues by using semaphores
  - Dynamically link global object with stacks at run-time

Functional Interface in CamlIDL

- CamlIDL
  - stub code generator
  - generates C stub code required for Caml/C interface based on IDL specification
- Neighbor Table
  - Write
    - *void write_ntable([in] struct ntable_entry ntable[], [in] int num)*
  - Read
    - *[int32] int getNeighborForInterface([in,int32] int intf)*

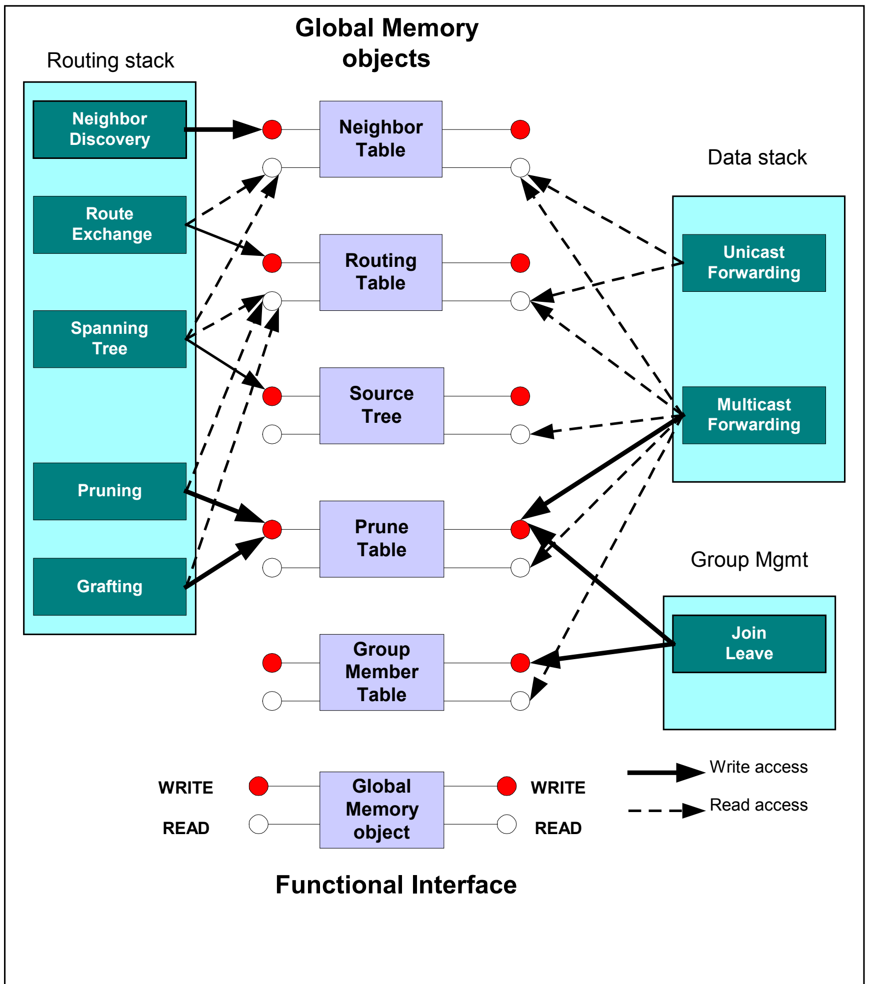
```
struct ntable_entry {
int32 intf_addr; // interface IP address
int32 nbor_addr; // neighbor IP address
};
```

Multicast Service – Operational Overview

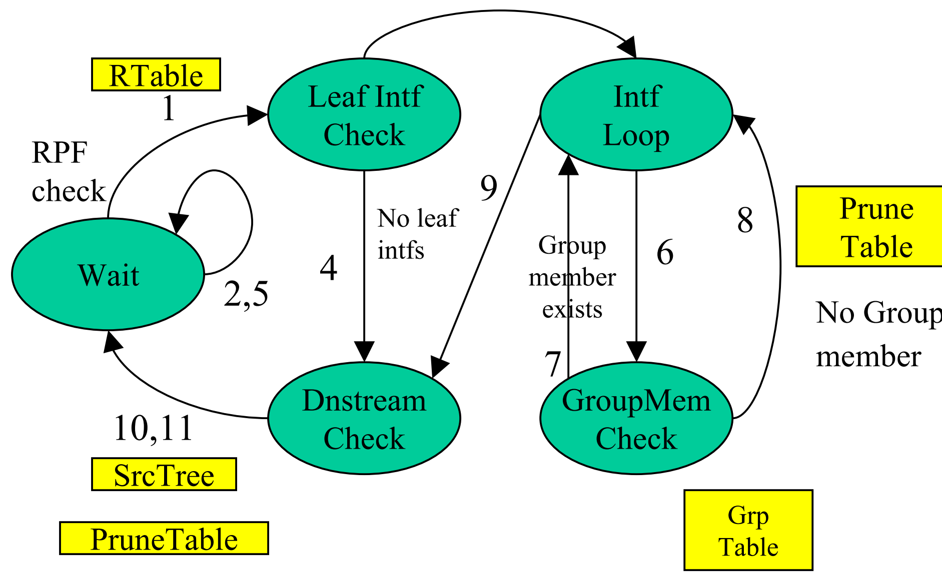


- Initialization
- Routing stack runs
- Neighbor Table updated
- Routing Table updated
- Source Tree updated
- H1, H2 joins group G1
- A multicasts to group G1
- H1 leaves G1
- H2 leaves G1
- H1 joins G1 again

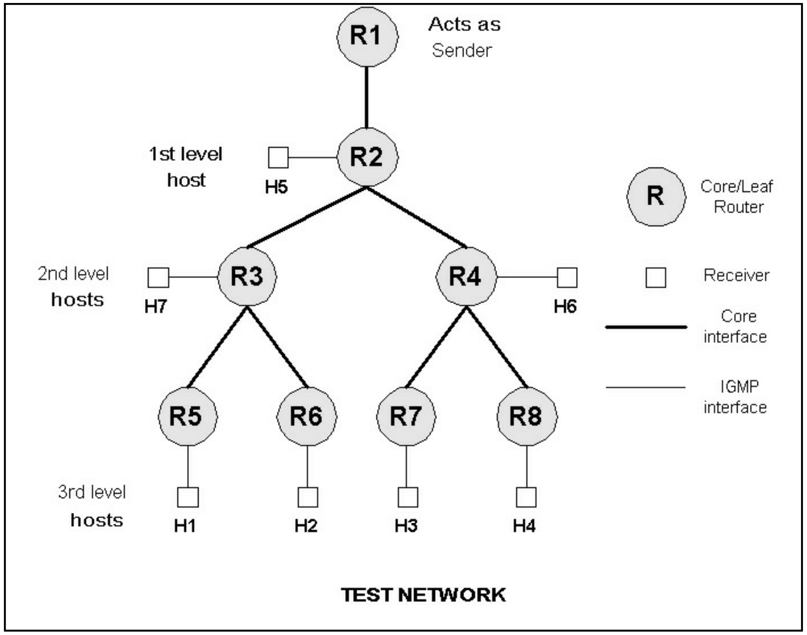
Protocol Interactions thru Global Memory



Multicast Forwarding - RSM



Performance Evaluation – Test Setup

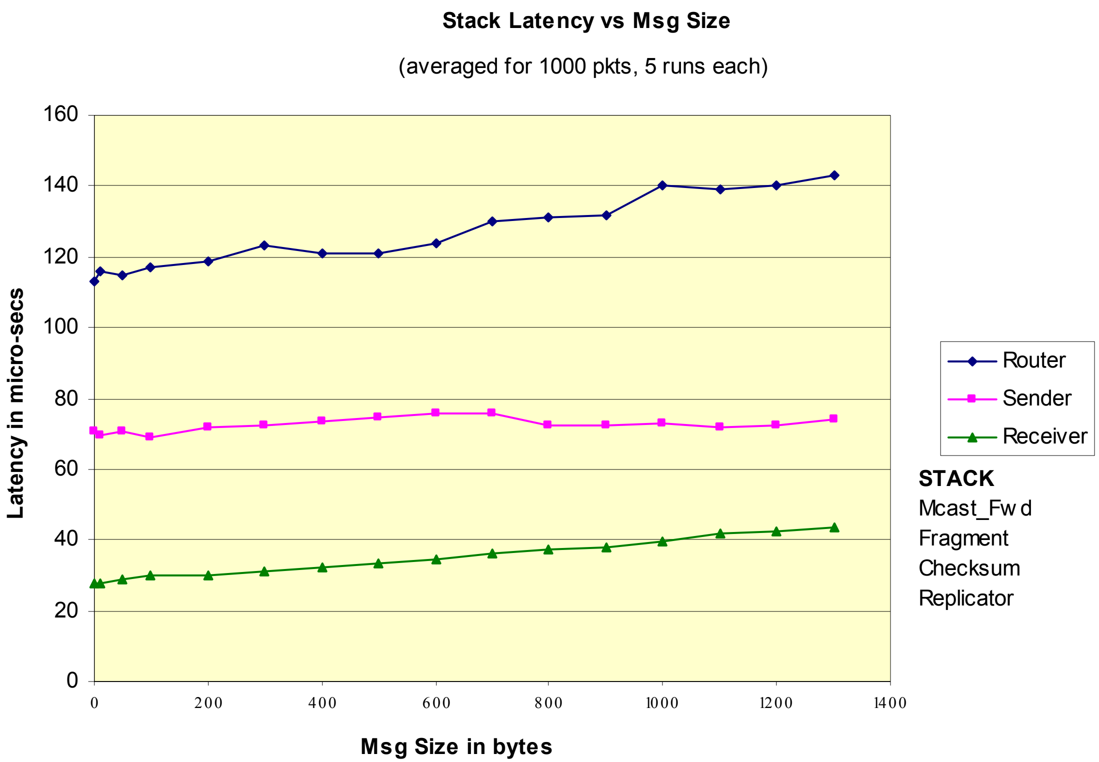


- Pentium III 800 MHz
- 256 MB RAM, 20 GB HDD
- 100 Mbps NICs
- RedHat 7.1, Linux 2.4.6
- OCaml v3.06, native code

- 15 node test network
- Ensemble test applications similar to ping, ttcp used
- Metrics
  - Stack/Component latency
  - One-way end-to-end latency
  - Basic Multicast Throughput
  - Reliable Multicast Throughput
  - Join/Leave latency

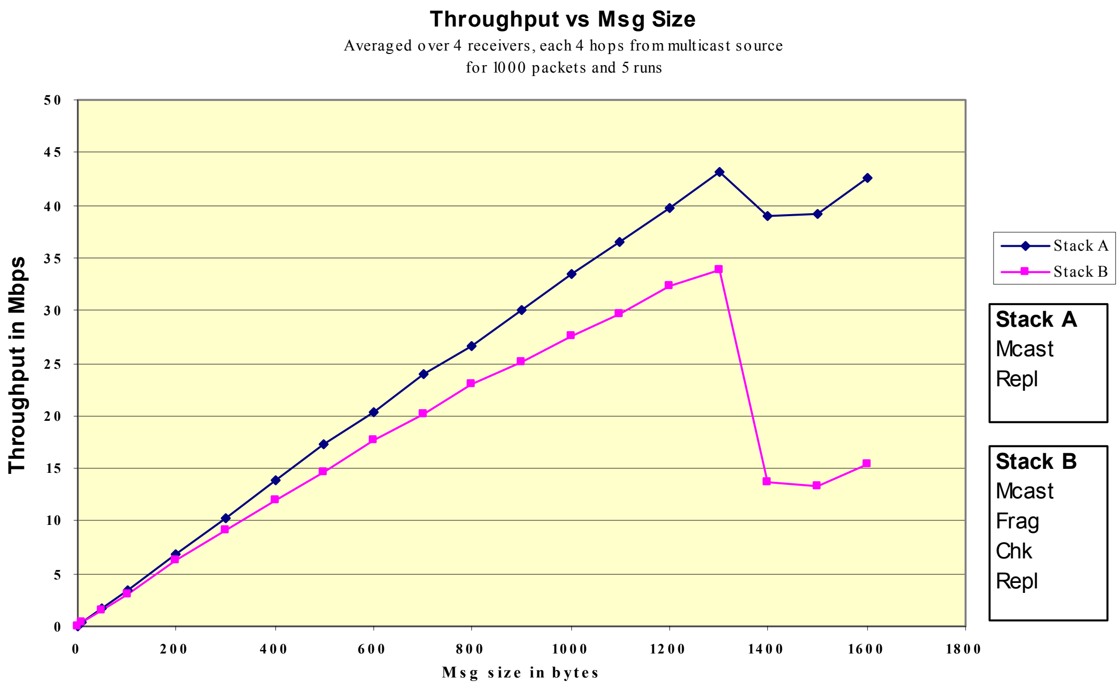
- Performance Improvement Factors
  - Native-code *ocamlopt* compiler instead of byte-code *ocamlc*
  - Reducing global memory lookups , use of caches
  - Order of guards

Stack Latency vs. Message Size



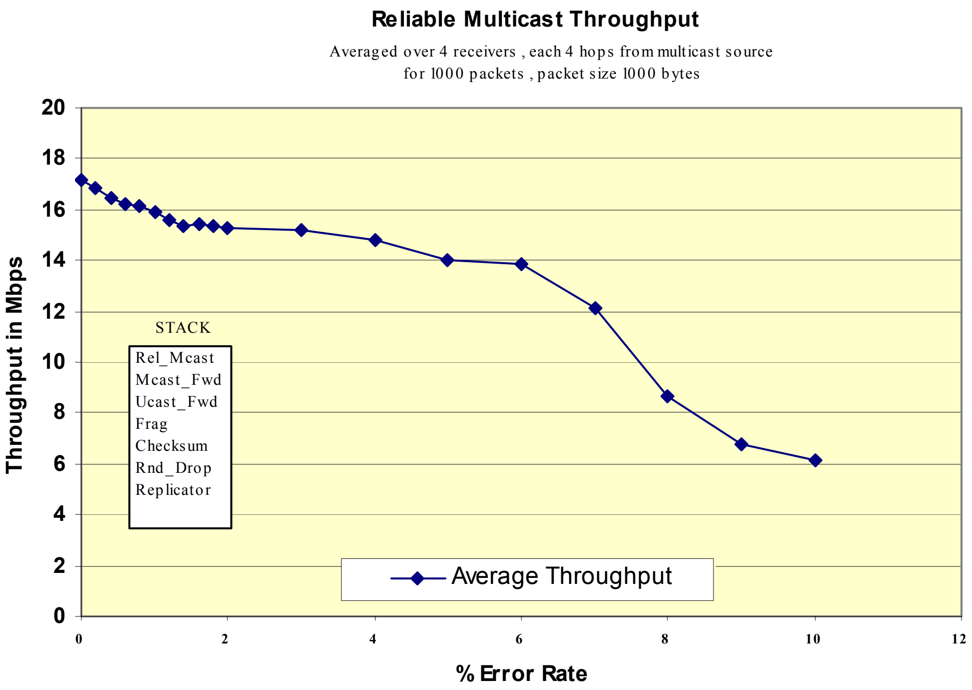
- 5 runs of 1000 pkts each
- Global memory lookup 1 in 100 pkts
- Stack latency increases with message size due to checksum component

Throughput vs. Message size



- 5 runs , 1000 pkts each
- Receivers 4 hops from source
- Sharp decrease after 1300 bytes due to fragmentation
- Stack A uses IP fragment
- Stack B uses *fragment* component

Reliable Multicast Throughput vs. Error rate

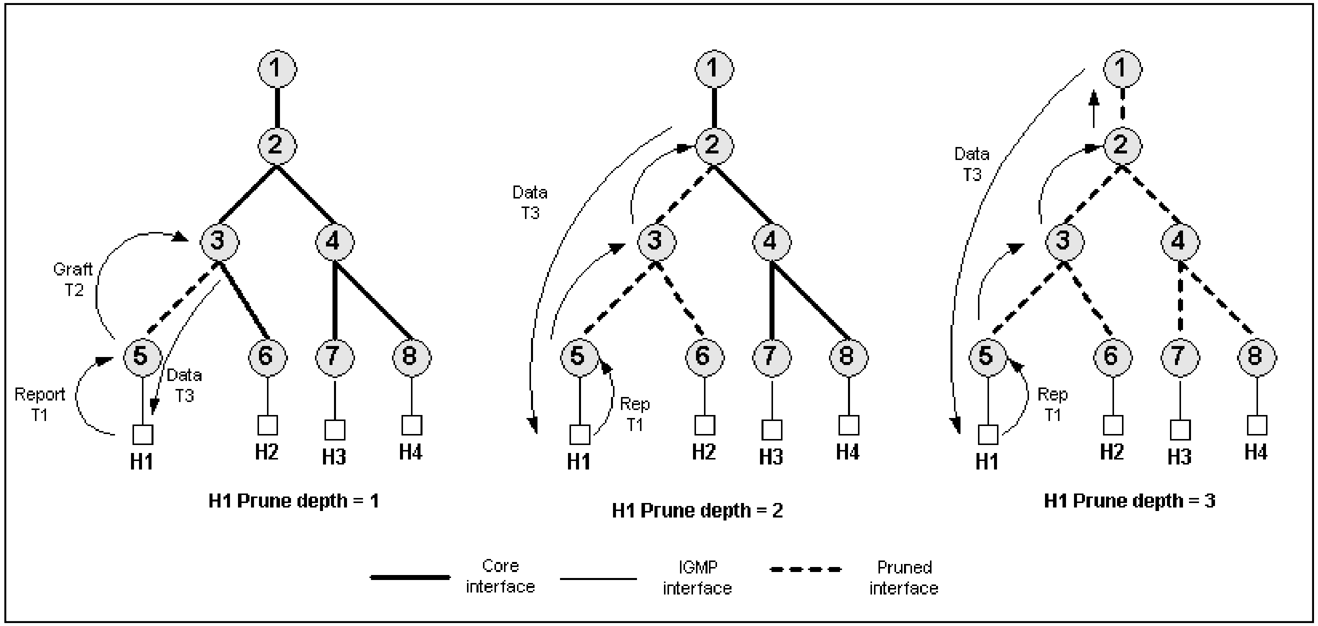


- 5 runs, 1000 packets each 1000 bytes
- Receivers 4 hops from source
- *Random_Drop component* simulated link-error rate
- Receiver-initiated NACK scheme used for *Reliable component*

Other metrics

Component Latencies					
	Msg Size	MCAST	FRAG	CHKSUM	REPL
Node	bytes	(in micro-seconds)			
Sender	1000	26.09	8.23	20.49	7.61
Receiver	1000	3.26	3.37	19.41	5.04

- Component Latencies at sender and receiver nodes
- Join and Leave latencies



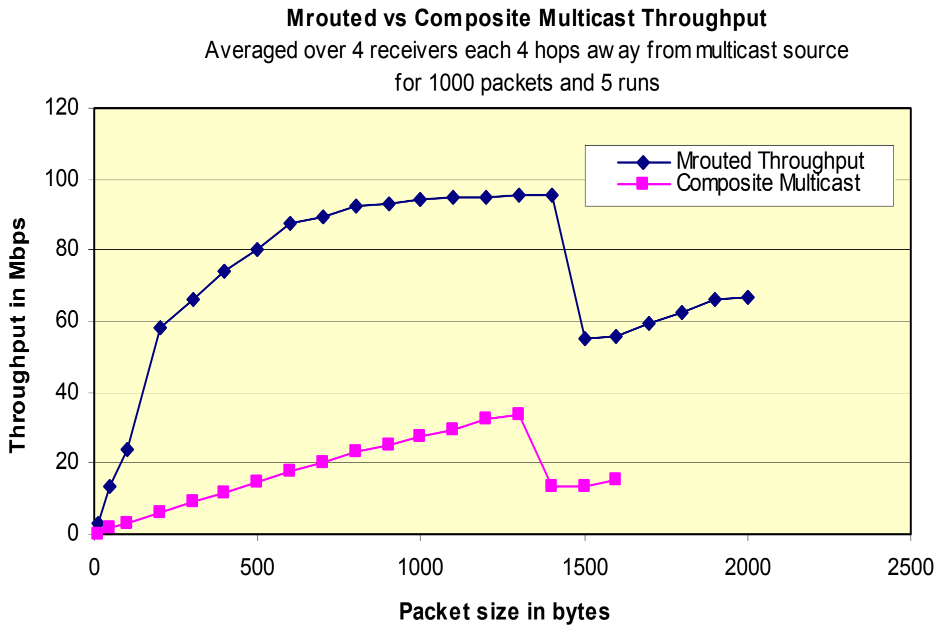
Timer	(seconds)
Query	0.1
Graft	0.1
Prune	0.1

Prune Depth	Join Latency (milli-seconds)
1	405
2	458
3	535

Leave Latency : 146 ms

Sender rate : 10 pkts/sec

Comparison with Linux IP Multicast



- Mrouted 3.9 evaluated on same test network
- *Iperf* tool used to measure end-to-end throughput
- Composite multicast just worse by a factor of 2-3.
  - SM execution adds overhead
  - Strict layering in framework prevents pointer arithmetic on buffers
  - Ensemble is a user-level program

Summary

- Novel approach for building network services from composite protocols
- Demonstrates applicability and feasibility of composite protocol approach to data-plane and control-plane protocols.
- Addresses challenging issue of inter-stack communication using global memory
- All components and global memory objects implemented and tested for both functionality and performance

Future Work

- The multicast service can be extended to support multi-point to multi-point model
- Implement complex multicast protocols like MOSPF/ PIM
- Security and network management protocols
- Improve performance
- Deployment of service on an active network

Questions ?