LIANE—Composition for Active Networks

Samrat Bhattacharjee Ellen Zegura

Networking and Telecommunications Group

College of Computing, Georgia Tech.

Atlanta, Georgia

Ken Calvert

Department of Computer Science

University of Kentucky

Lexington, Kentucky

http://www.cc.gatech.edu/projects/canes

Sponsor: DARPA

Active Networking

Active networks provide a *dynamically programmable* **user-network interface**

- *Users* supply both *programs* and data
- Network nodes both forward data and execute user programs

Benefits:

- Rapid and unilateral deployment of new protocols and algorithms
- Mechanism to exploit application-specific **and** network-layer information

Service Composition in Active Networks

- Network exposes a *programming interface*
- An “underlying program” executes at each node
- Users *inject* computations into the network

Questions:

- How does the injected program “bind” to the underlying program?
- How is the interaction between the underlying and injected programs controlled?
- Is it possible to reason about the composite computation?

Answer: It depends. . . on the programming interface and supported programming model.

The Processing Slot Programming Model

A general programming model for active networks:

- **Underlying program** encapsulates uniform per-packet processing and is resident at each node
- Underlying program identifies **Processing Slots**

Processing Slots:

- Slots specify **when and where injected programs** may execute
- Underlying program **raises** a slot iff slot-specific conditions hold
- Injected programs **bind** to specific slots

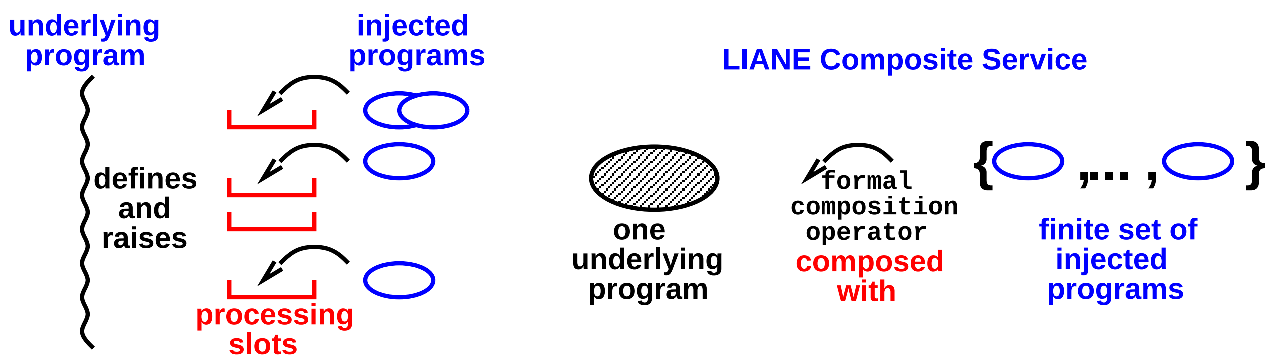
Multiple injected programs may bind to a single slot

All programs bound to a slot execute concurrently when the slot is raised

Reasoning about Active Networks

LIANE Language Independent Active Network Environment:

- A formal model in UNITY notation and logic
- Underlying programs identify resource bounds, restrictions and obligations of injected code for each slot
- Transformation technique to form composite



Injection preserves all properties of underlying programs

Unless resource bounds are violated, properties of injected algorithm are preserved as well

LIANE Example: Generalized Forwarding Function

Parse packet, obtain source s, destination d

⟨**Slot 0:[null]**⟩ {trace route, caching, select route table, discard}

i := **Lookup**(d, `route table` R)

if $i = \perp$ then ⟨**Slot 1:[null]**⟩ {error messages to source}

⟨**Slot 2:[null]**⟩ {send route back, select alternate interface}

if i is congested then ⟨**Slot 3:[discard]**⟩ {cong. control algorithm}

else ⟨**Slot 4:[null]**⟩ {scheduling algorithm}

enqueue packet for i.

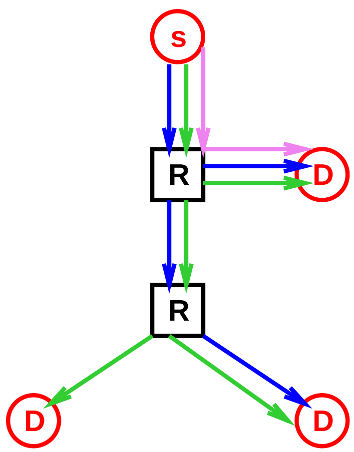
Figure 1: Example Underlying Program with processing slots

Example Injected Program: **Mobility, Congestion Control Algorithms**

Application: Network Support for Multicast Video

Multicast video over best effort networks:

- Three locations for adaption: Sender, Receiver, *Network*



Receiver-based adaption:

- Media is partitioned into layers
- Receivers join different multicast groups corresponding to video layers
- Receivers adapt to network conditions by joining and leaving different multicast groups

In-network adaptation:

- Media-specific reduction techniques installed in routers

Overview of Experiments

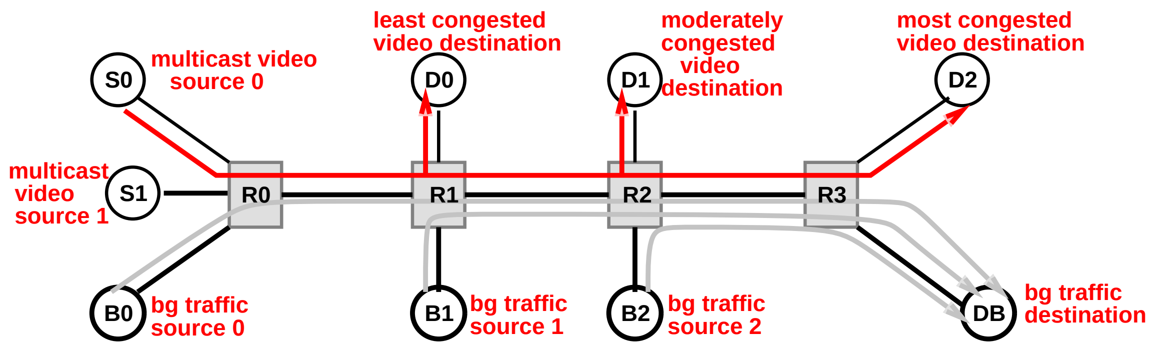


Figure 2: Experimental Topology

Simulation experiments using ANSWER:

- All multicast group actions were instantaneous
- MPEG video simulated: 3 layers, 615 Kbps, 30 fps
- Experiments with different:
  - background traffic scenarios, number and priority of sources, decoding schemes, capabilities at routers

Multicast Result: Single Video Source

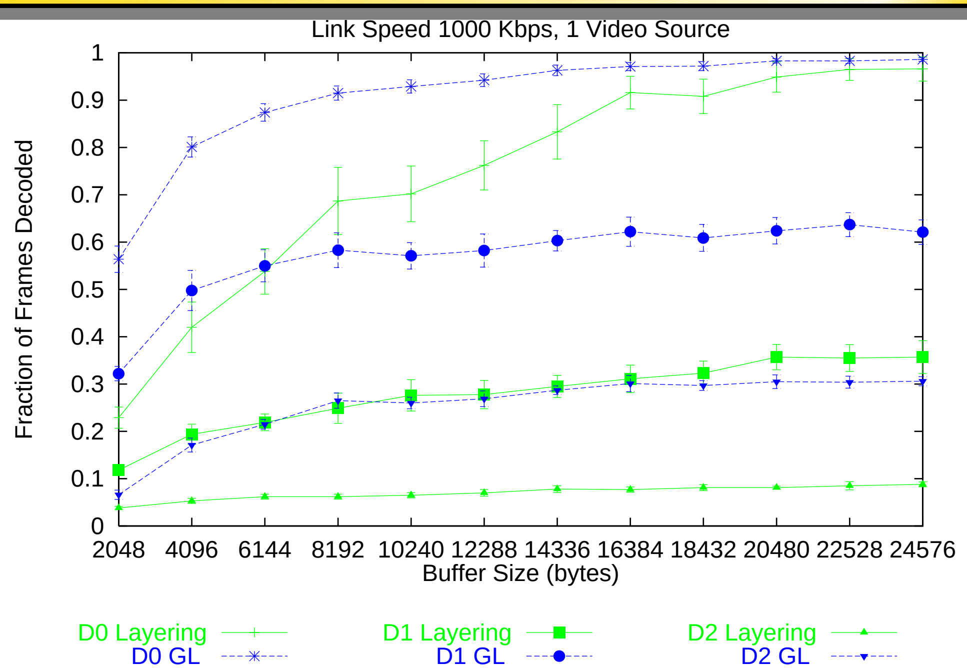


Figure 3: Fraction of frames decoded with varying buffer size at routers

- For uncongested destinations, receiver-based adaptation requires larger buffers to provide similar performance
- For congested destinations, network-based adaptation provides 2-3 *times* better performance

Detail Snapshot: Single Video Source

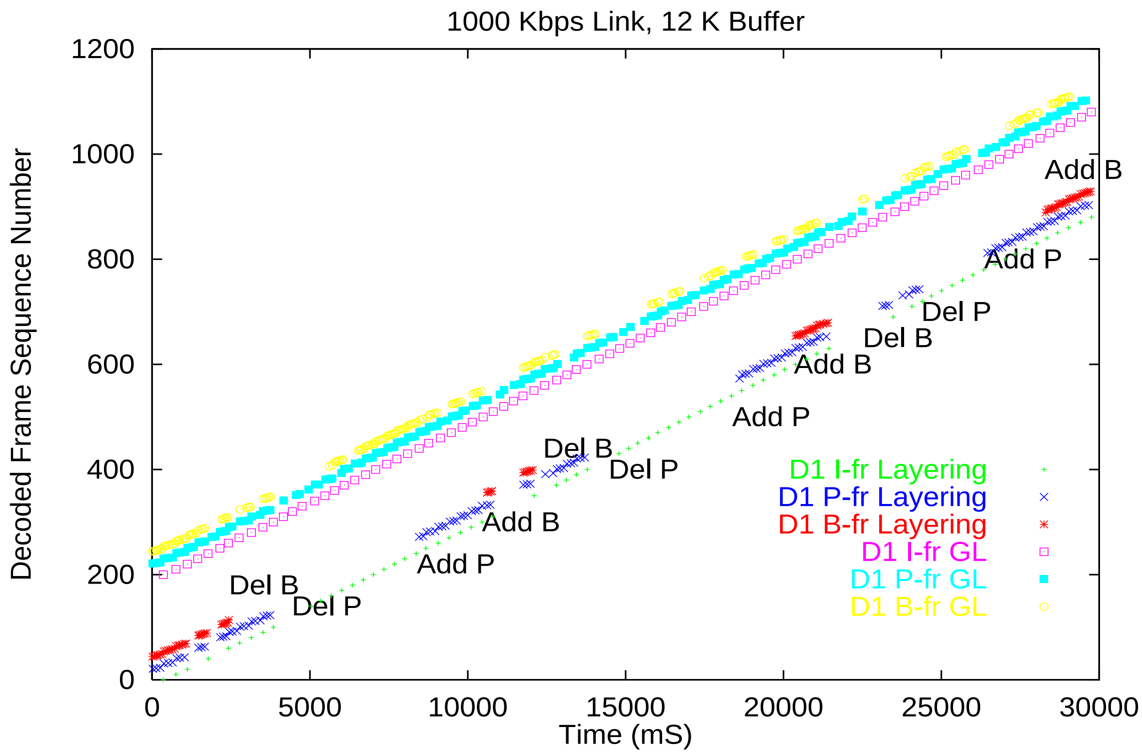


Figure 4: Sequence of frames received at receiver D1

Multiple Sources with Priority

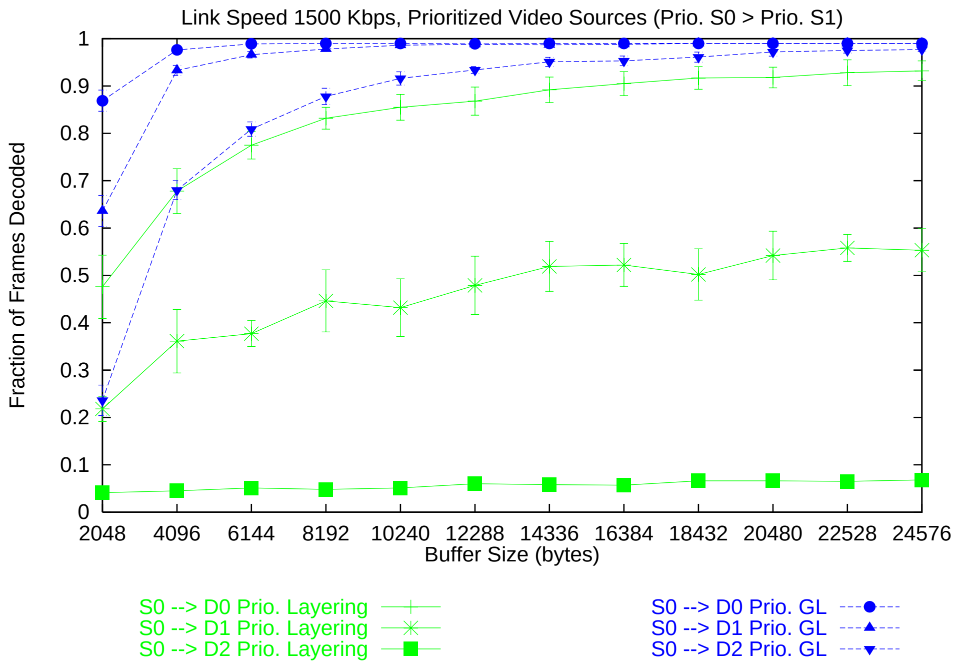


Figure 5: Fraction of frames decoded for high priority source

- Difficult to eliminate effects of the lower priority sources under receiver-based adaptation — join experiments of lower priority sources disrupt the high priority traffic

Citizenship

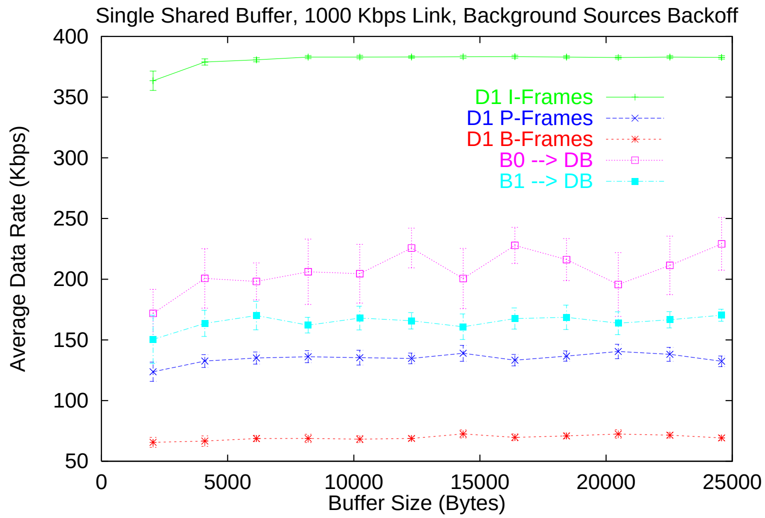


Figure 6: Single Shared Buffer

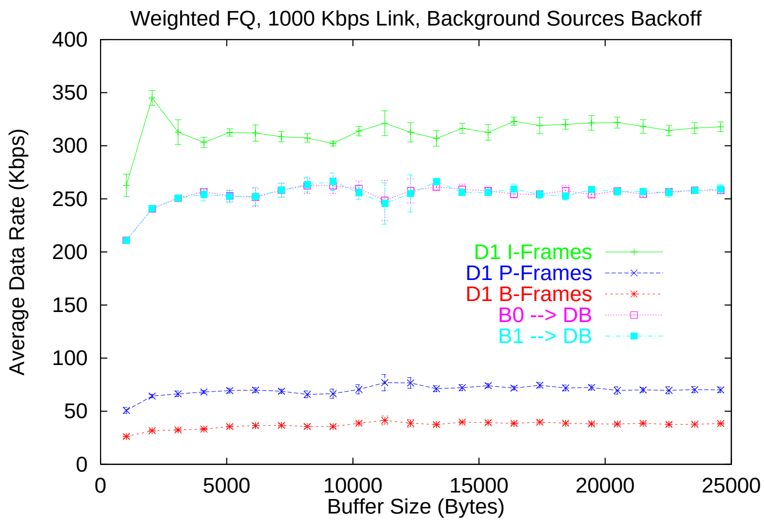


Figure 7: Weighted Fair Queue

- Bandwidth is shared unfairly with in a shared buffer
- Weighted fair queuing *equalizes* the bandwidth allocation

Current Work

LIANE

- Discrete-event simulator ANSWER implemented using the slot model
- ANSWER allows experimentation with slot-processing model
- Implementation of LIANE

Applications

- Work on other applications: Virtual Topologies, Anycasting
- Larger topologies
- “Partially active” networks