

# Programming the Internet

## Overview: Questions about Programming the Internet

Ken Calvert, U. of Kentucky, USA

### Why?

Risky Business: Developing an Infrastructure for Third-Party Computation

Ian Wakeman, U. of Sussex, UK

### How?

Rule-Based Systems Programmability

Michael Smirnov, Fraunhofer FOKUS, Germany

### What?

Analog Programs: Mobile Code for Fibers and Ethers

Christian Tschudin, U. Basel, Switzerland

Mapping Data-path Functions to Network Processor Configurations

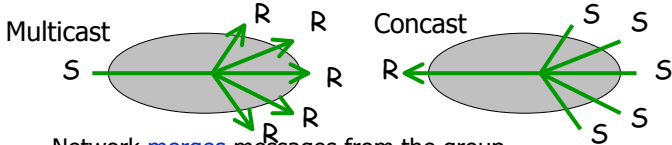
Hao Che, U. Texas-Arlington, USA

## Question: Why?

### Why do we want a programmable Internet?

- To speed evolution, overcome “ossification”  
But stability of basic processing is crucial for the fast path
- To customize processing along forwarding path  
But what do you really need, beyond forwarding/scheduling?
- To improve scalability of group applications  
Example: Concast
- To overcome limitations (info hiding) of the best-effort service abstraction  
Example: Using Ephemeral State Processing to identify branch points in multicast trees

# Concast

- Scalability through [abstraction](#)
    - Inverse of multicast
      - Single address represents an arbitrary number of senders
- 
- Network [merges](#) messages from the group
    - According to user-supplied [merge specification](#) (=program)
  - Benefits both receiver and network
    - [Multiple sends result in a single message delivery](#)
    - Reduced bandwidth requirements
  - Merging happens only where required (on direct path to R)

## Question: Why?

### Why would providers want a programmable Internet?

- [Because users will pay to get it](#) (see also Wakeman)

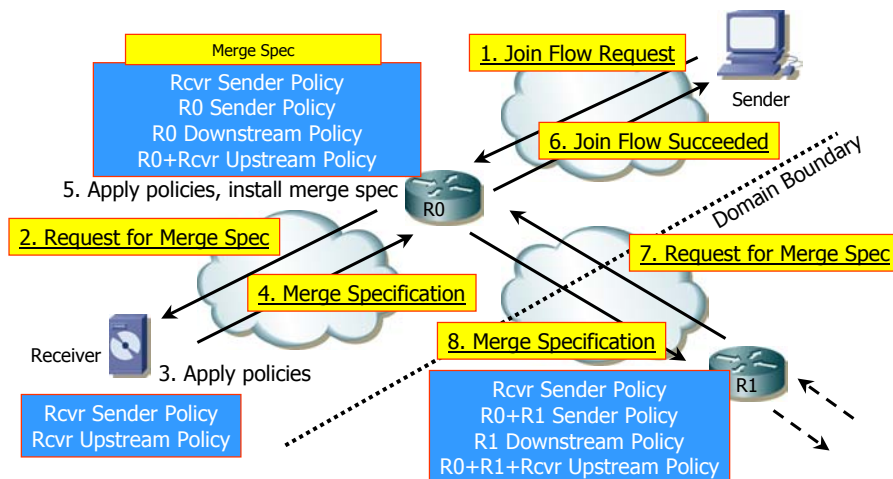
### (Why) should users trust shared infrastructure to:

- Forward packets to specified destinations?
- Reserve end-to-end bandwidth for their packets?
- Process user data to improve scalability?
  - [Example](#): multicast feedback aggregation
- Process content en route from content provider?
  - [Example](#): transcoding news video for low-bandwidth links
- [Enforce user policies](#)?
  - [Example](#): [controlling concast group membership](#)

# Concast Security Policies

- **User (Receiver)**
  - Concern: Data integrity, authenticity, confidentiality
  - Application-level policy: Which senders can participate
  - Network-level policy: Which **routers** (domains) can participate in the flow (i.e. be upstream)
    - Perform merging
    - **Enforce policies!**
- **Provider (Router)**
  - Concern: Only paying customers get access to the service
  - Network-level policy: Which entities can participate as senders/receivers
  - Network-level policy: Which **routers** (domains) can be downstream/upstream

# Concast Policy Monotonicity Requirements



## Question: How?

### How should the network be programmed?

- Lots of possible answers, we have only begun to explore this (see also Smirnov, Tschudin)

### How to charge for a programmable Internet?

- At signaling time, not forwarding time
  - Untrusted → trusted transformation, policy application too costly for data plane
- Locally, not end-to-end
  - At least two providers involved in end-to-end service
  - Avoid multilateral settlement protocols, transitive trust

## Question: How?

### 2-level model of programmability

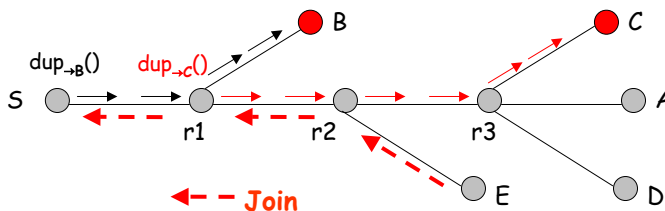
- Node-local functions enabled directly by user
  - Examples: duplication, redirection, dropping
    - Controlled via secure signaling protocol
    - Affecting only packets "belonging to" the paying user
    - Paid for at signaling time via [bilateral agreement](#) between end user and node owner
- End-to-end functions available to all packets
  - IP-like resource requirements
  - Fixed forwarding-time computations, close to fast path
  - Sequence per-packet computations to form global

# Lightweight Processing Modules and Ephemeral State Processing

- LWP: fixed-function per-flow modules
  - Specified by flowspec, ...
  - Soft-state
  - Examples: duplication, redirect, dropping
- ESP: allow packets to create, manipulate small amounts of state in routers
  - Fixed instruction set; one instruction per packet
  - Per-packet processing, storage requirements bounded due to state lifetime
  - Narrow interface to forwarding function: drop or forward

# Implementing Multicast with ESP+LWP

- LWP "dup()" function installed by receivers to duplicate and forward marked packets to themselves
- To join the tree:
  - Discover closest existing branch point (via ESP)
  - Activate a dup() to self there
  - Find optimal branch point (via ESP), move dup() there

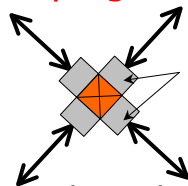


# Programmability via LWP and ESP

- Deployment strategy
  - Recover costs via LWP
    - Value-added: end-to-end services like multicast
  - Deploy ESP to enable use of LWP
- End-to-End Services
  - Multicast
  - Layered multicast congestion control
  - Open issue: what others?
    - Can QoS be done with custom processing at one or two nodes? Are congestion-location/timescales consistent with a LWP-based approach?

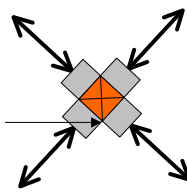
## Question: What?

### What to program?



- Global fault-tolerance
- Run at channel speed
- Simple interface to forwarding path

Do processing here...  
... not here



- Not fail-safe
- Maintain forwarding state
- Run at interconnect speed = (wire speed)<sup>n</sup>

How to structure services as channel computations?  
(See also Tschudin, Che)

# Programming the Internet: Open Issues

- Why:
  - What services will users pay for?
    - Is BEUF special?
  - Trust acquisition, third-party policy enforcement
- How:
  - Integrated programming models
  - Can we do QoS via local customization (at the right location)?
- What:
  - Mapping services onto channel-based computations