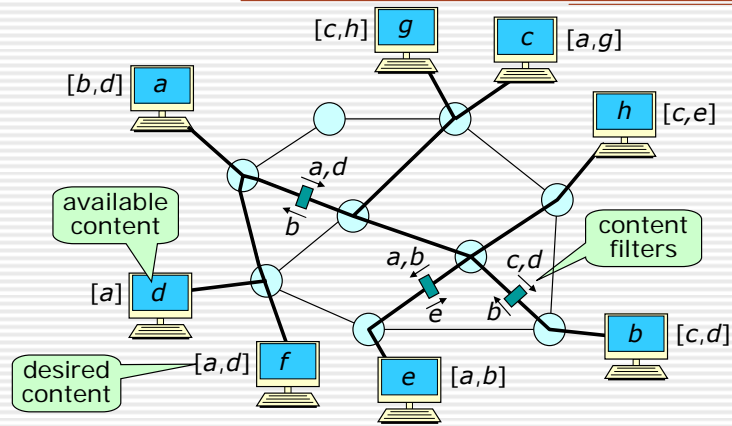


# Building Metanets

- Metanet for distributed interactive simulation
- DoS-free metanet with datagram service
- Video narrowcasting metanet
- Large-scale science metanet

# Scalable Distributed Simulations



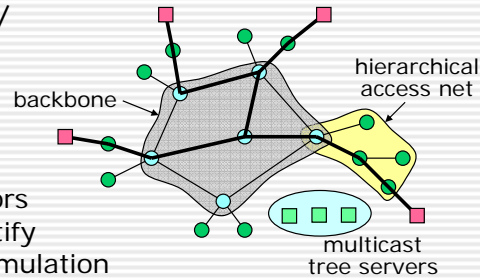
- Informed update propagation for distributed simulation.
  - » many-to-many multicast channel
  - » interest filters limit update propagation
- From Zabele, et. al. [2001].

## Design Issues

- Application environment
  - » simulations involving large numbers of distributed users
  - » many separate simulation sessions operating simultaneously
  - » well-connected end systems with ample computation & storage
  - » real-time interaction requires low delay, consistent views
  - » applicable to training and online gaming
- Objectives
  - » scalability – many sessions ( $>10^5$ ), many users/session ( $10^4$ )
  - » excellent performance – low latency, consistent views
  - » use smart multicast to distribute data where needed
- Performance implications
  - » session with 5,000 users, 50 ms update interval generates 100K updates/s
    - so  $\approx 100K$  packets/sec,  $\approx 200$  Mb/s
    - rates remain tolerable at 50,000 users
  - » speed of simulation entities – impact on interest filters
  - » overhead due to join/leave dynamics
    - if avg connection time of 15 minutes then 6 enter/leave each sec

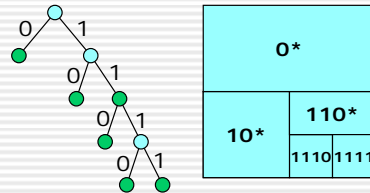
## Multicast Tree Construction

- Support dynamic addition/removal of endpoints.
- Set of multicast tree servers record tree info.
  - » servers known to all metarouter control processors
  - » use DHT techniques to identify tree server(s) for a given simulation
  - » when new user joins simulation
    - retrieve *waypoint* node and bandwidth estimate from tree server
    - take unicast path from user to waypoint & connect on intercept
- Metanet backbone topology known to multicast servers.
  - » including link lengths and capacities
  - » backbone fairly static and not too large (100 nodes or less)
  - » user's metanet address identifies its backbone node
- Multicast servers adjust link bandwidth reservations as simulation runs – may re-route if necessary



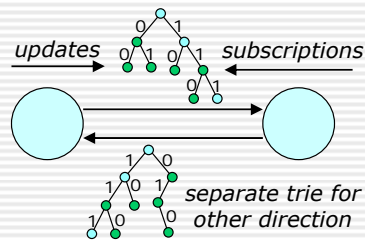
# Location-Based Filtering

- Central element of infrastructure.
  - » must be broadly applicable to different simulations
  - » should facilitate efficient filtering on specific locations or large regions of space
- Binary trie representation
  - » apply to 2d, 3d or higher dimensional spaces
  - » separate trie for each simulation
    - simulation decides interpretation
    - max trie depth may vary
- Packet filtering
  - » at each router, broadcast to all outputs in multicast tree, then filter
  - » *propagate filter* associated with *location prefix*
  - » state update packet specifies location of simulation entity
  - » packet forwarded if one or more propagate filters on trie path



# Filter Maintenance

- Each link has trie containing endpoint locations & filters
  - » locations of endpoints on "left side" of link placed in trie
    - inserted as side-effect of state update packets from left
    - removed when "stale" (simulation-dependent parameter)
  - » subscriptions from endpoints on "right side"
    - inserted by subscription packets, removed when stale
  - » update packet from left is propagated if
    - there is a subscription for region containing endpoint location
    - or, endpoint location is not previously stored in trie or is stale
  - » subscription packet from right is propagated if
    - there are endpoints stored in trie for region of interest



## ■ Alternate approach

- » divide tree into central *core* and periphery
- » propagate updates to all core nodes
- » propagate subscriptions to first core node

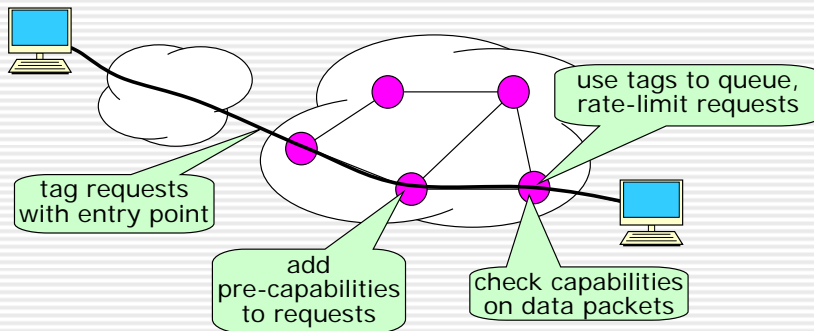
## Update Synchronization

- For interactive applications, important to deliver updates to users at same time.
- Access metarouters maintain clocks synchronized to within 1 ms.
- State update packets timestamped on receipt by first access router.
- Packets ordered by timestamps at outgoing access link queues & delayed until ( $now - timestamp = target\ delay$ )
  - » target delay chosen to be slightly larger than max propagation delay and max expected queueing delay
  - » for highly interactive simulations, limit to 50 ms
- Late packet counts reported to multicast tree server
  - » may trigger re-organization of tree, if cannot fix by increasing bandwidth reservation

## Performance Issues

- Primary objective is to limit traffic to endpoints, while ensuring all relevant updates reach them
- Multicast tree maintenance
  - » bound maximum delay and use system resources efficiently
    - reserve enough bandwidth to minimize queueing
    - delay determined primarily by propagation time
  - » monitor bandwidth usage on each link and adjust reservations as usage varies – maintain safety margin, delay reductions
  - » backbone restructuring
    - find alternate path joining backbone nodes on opposite sides of congested tree section
    - replace most congested link with new path
- Filter update processing
  - » trade-off between data forwarding and processing overhead
  - » related to speed of location changes relative to updates
    - if location is area 10 meters across, and simulation entity can move at 100 km/h, takes about 7x50 ms to change locations
- Internal metarouter bandwidth usage

# DoS Attack Mitigation



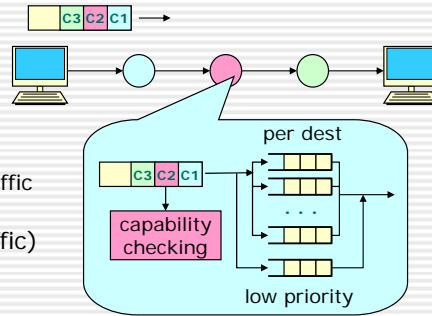
- Must get receiver's permission before sending to it.
- Routers block transmissions lacking *capability*.
  - » routers each create piece of capability that only they can check
- Request rate limited by queueing based on entry point.
- From Yang, Weatherall and Anderson [2005].

## Background

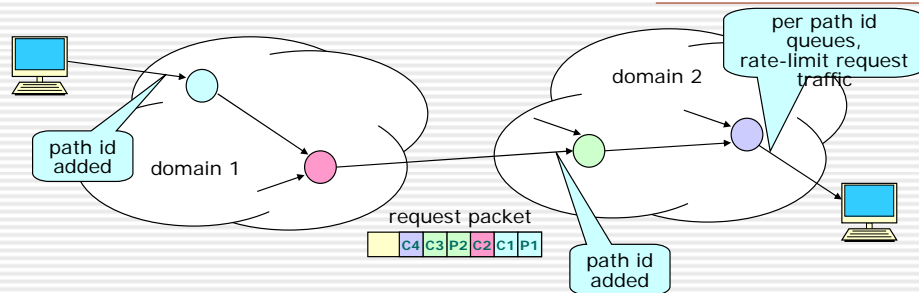
- Problem: protecting against Denial of Service attacks
  - » datagram networks are intrinsically vulnerable to DoS
  - » attacks have become common and impose significant costs
- Current (partial) solutions
  - » source address filtering
  - » traceback
  - » overlay filtering
  - » pushback
- Proposed Traffic Validation Architecture
  - » comprehensive approach based on *capabilities*
  - » senders must obtain permission from receiver before sending
    - permission represented by capabilities (hard to forge tokens)
  - » routers check capabilities and forward non-compliant traffic at lower priority
  - » other mechanisms protect request channel from DoS
  - » caching used to reduce cost of capability checking

# Overview of TVA Data Forwarding

- Packets sent with capability for each router along path.
- Routers check capabilities.
  - » compliant packets placed in per destination queues
    - keep excessive traffic to subverted destination from blocking “good” traffic
  - » non-compliant packets placed in low priority queue (with legacy traffic)
- What’s in a capability?
  - » router  $i$  contributes pre-capability
    - local timestamp  $t_i$
    - $\text{hash}(t_i, \text{src adr}, \text{dest adr}, \text{secret})$
    - secret changes twice per timestamp rollover period
    - use hash function that is hard to invert
  - » destination converts pre-capabilities to capabilities
    - $\text{hash}(N, T, \text{pre-capability})$ , where  $N$  is byte count,  $T$  is time limit
  - » packets also include plain-text versions of  $N$  and  $T$
- Routers retain state to verify byte count.



## Obtaining Capabilities



- Senders use request packets to obtain capabilities.
  - » piggy-back on TCP SYN packets to avoid extra RTT
- To prevent DoS attack using requests,
  - » path ids (hash of entry id) added at trust boundaries
  - » request packets queued based on path ids
  - » requests forwarded at fraction of link rate (e.g. 10%)
- Ensures that requests don't overwhelm receiver and that most legitimate requests get through.

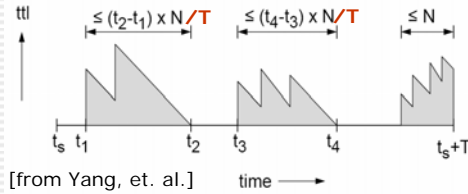
## Bounding State

- Most functions require no per flow state in routers.
  - » to check capability, compute  
 $\text{hash}(N, T, \text{hash}(t_i, \text{sadr}, \text{dadr}, \text{secret}))$   
most required fields  $(N, T, t_i, \text{sadr}, \text{dadr})$  present in packet and only two secrets needed per router
  - » check that capability has not expired by comparing  $T+t_i$  to current time
- To check that sender has not sent too many bytes, need record of number sent previously.
  - » creates possibility of attack designed to exhaust router state
- To bound state required,
  - » when router  $i$  receives packet from flow  $j$  with length  $L$ ,
    - if no flow state, create pair (byte count, state exp. time= $\text{now}+LT_j/N_j$ )
    - if state present, add  $L$  to byte count (and check against limits) and increase state expiration time by  $LT_j/N_j$
  - » if more space needed for state, remove state pair for which expiration time has been reached

## Bounding State (continued)

### ■ Enforcing flow limits.

- » if flow state was created at  $t_1$  and removed at  $t_2$ , then flow sent at most  $(t_2 - t_1)(N_j/T_j)$  bytes during interval



- » so consider sequence of periods in which flow state is present for flow  $j$ , with the last period ending before capability time limit
    - number of bytes sent during these periods is at most  $N_j$
  - » if state expires just before capability time limit, can send at most  $N_j$  more bytes
- Given minimum rate  $(N/T)_{min}$  for capabilities, an input link with capacity  $C$  requires state for at most  $C/(N/T)_{min}$  flows (for 10 Gb/s link, 100 kb/s min rate, need 100K flow state records)

## Reducing Packet Overhead

- Capability header adds at least 8 bytes to packet header for each router on path.
- Can reduce overhead by caching capabilities.
  - » sender includes random nonce with packets
  - » routers cache relevant capability information and nonce
  - » after first packet sent, sender includes only nonce with packet
  - » router checks nonce against value stored with per flow data; if nonce matches, it does resource checks and updates state
- If packet is received with nonce, when no flow state is present, it is forwarded at low priority
  - » senders can prevent this by sending full capability when cache state has expired
  - » senders maintain state expiration time using same algorithm as router – send full capability for packets with expired state
  - » flows sending below their nominal rate may need to include capabilities with each packet – expensive for short packets

## Limiting Impact of Route Changes

- Route changes invalidate capabilities.
  - » packet takes path different from one for which capability was constructed
  - » routers mark packets with invalid capabilities and forward at low priority
- When destination receives packet with invalid capability,
  - » it marks bit in next return packet sent, informing sender that it needs to request new capability
  - » sender then issues request packet, triggering construction of new capability by routers on new path
- Requires route changes to be relatively infrequent.

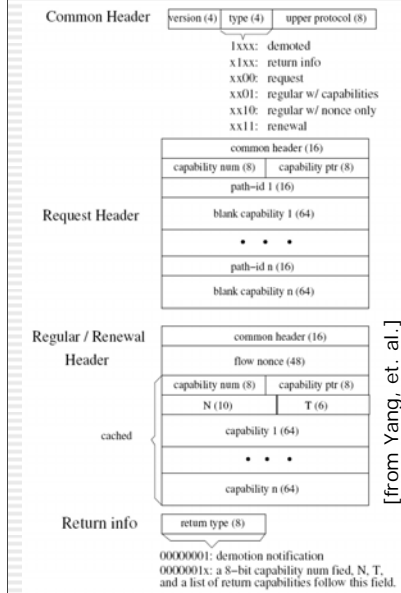
## Limiting Cost of Fair Queueing

- Fair queueing on destination address prevents receivers from getting unfair share of link bandwidth.
  - » but, this requires separate queue per receiver
- To limit cost of queueing
  - » maintain separate queues only for flows with per flow state
  - » remaining flows placed in separate shared queue
- Ambiguities in text
  - » appears to advocate use of per-flow queues, not per-receiver queues,
    - enables bandwidth hogging using single receiver, many spoofed source addresses
    - correct by using per receiver queue whenever some flow for that receiver is in cache – requires reference count per receiver queue
  - » does not address packet ordering issue raised by queue switching

## Short, Slow or Asymmetric Flows

- TVA most efficient for long, higher rate flows.
- Short flows can be quite inefficient (e.g. DNS queries).
- Effect on aggregate efficiency is small.
  - » assuming that most traffic carried by longer, high rate flows
- Ratio of request traffic to data traffic may be different for different links.
  - » network operators must determine appropriate ratio for the traffic mix on a given link
  - » extreme example: link leading to root DNS server will mostly carry request traffic

# Implementation

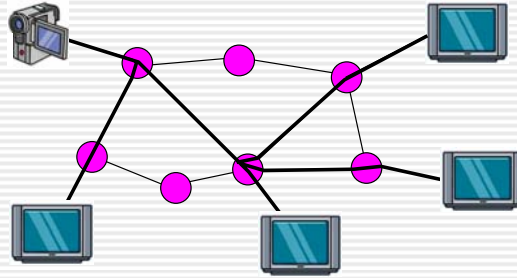


- Shim header that precedes IP header.
- Request packets
  - » path id per domain (16 bits)
  - » pre-capability per router (64 bits)
    - eight bit timestamp plus hash
- Data packets with cap.
  - » flow nonce (48 bits)
  - »  $N, T$  (16 bits) – units?
  - » capability per router (64 bits)
- Data packets with no cap.
  - » flow nonce (48 bits)

## Security of TVA

- Using another host's capabilities.
  - » to be effective, must share same path to destination
  - » such attackers are indistinguishable from legit. host, so host suffers
- Forging capabilities.
  - » strong cryptographic hash functions, key changes every 128 seconds
  - » effective attacker must break hash in  $\ll 128$  seconds
- Discovering pre-capabilities by triggering their return in ICMP error messages.
  - » allows sender to substitute different  $N, T$  values
  - » block by using packet formats that do not place pre-capabilities in first eight bytes of the packet returned by ICMP error message
    - apparently assuming ICMP applied to TVA packets, as with IP packets
- Compromised router masquerading as receiver (or receivers).
  - » attacker sends requests to colluding router, which returns capabilities
  - » affects upstream traffic, but downstream traffic is best-effort
  - » can crowd out traffic to receiver on congested upstream links
- Attacks on resources at capability routers
  - » can be provisioned for worst-case, independent of attacker behavior

## Video Narrowcasting



- Allow end-users to distribute real-time video.
  - » high school basketball games, birthday parties, conference, . . .
- Requires network support for multicast and QoS.
  - » allow "owner" to control access
  - » video format translation in network
- Directory services so users can find programs/friends.
- Pay for by advertisement – ad insertion by routers.

## Service Model

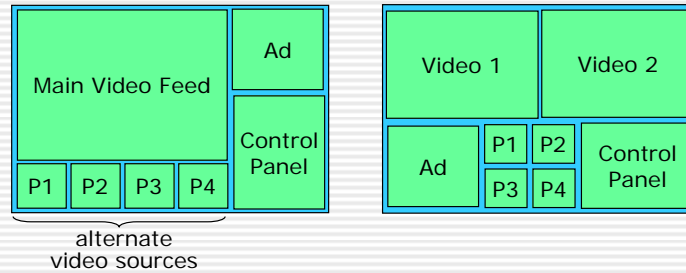
- **Viewers** connect through web-style interface
  - » “programs” have names and may require passwords
  - » to view program, enter name and password in form
  - » viewer may select payment options
    - free (with ads), low-pay (some ads), high-pay (no ads)
  - » directory of public programs also provided
  - » select view, when multiple cameras available
  - » manipulate camera pan/zoom when allowed
  - » when viewing program can get list of “friends” also watching
    - support multi-way audio conference among groups of friends
- **Additional options for *transmitters***
  - » create program and set payment options
    - may choose to pay for ad-free transmission, or free for selected viewers
  - » add cameras to program and set viewer-control options
  - » add audio streams from “announcers”

## Multicast Tree Maintenance

- Program may have several video sources and multiple audio channels
  - » one-to-many multicast for video
  - » all-to-all groups for audio with distributed audio bridging
- Users may choose to receive any video feed (or more than one), and setup audio groups with friends
- To simplify adding/removing users, route distinct channels on same multicast tree
  - » reserve bandwidth for all video feeds & several audio channels
- Each tree managed by a multicast tree server
  - » assign trees to servers using DHT methods
  - » servers know backbone topology
  - » when viewer joins tree, first get *waypoint* from tree server, then take unicast route towards waypoint – stop-on-intercept
- Propagate video feeds to all backbone nodes, selectively propagate to access nodes and users.

# Video Processing

## ■ Format options



- Create preview of each video feed near source
  - » I-frames only, quarter resolution (greatly reduced bandwidth)
- Feed ad stream into multicast tree from ad server
- Create single composite view at access router
  - » select primary video sources, based on user controls
  - » tailor to bandwidth of access link, resolution of viewing device

# Audio Processing

## ■ Announcer channels

- » may be multiple announcers on one channel (play-by-play and commentary) and/or multiple channels (alternate languages)
- » announcers on same channel hear each other
  - viewers only listen on these channels
- » announcers always transmit, signals combined at merge points in tree (add audio samples, with AGC to limit volume)

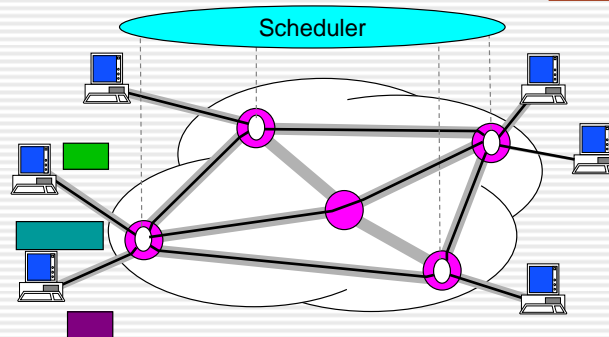
## ■ Friends channels

- » for natural conversation, allow anyone to speak at any time
- » but, if many friends sharing same channel, too much noise from open mikes
- » switch audio sources on/off as speech is detected (ramp on/off)
  - keep one source on at all times (most-recent speaker)
- » metarouters combine audio signals for up to 3-5 speakers
  - if too many sources present, select highest priority
- » access router turns off audio on silence after random delay
  - stay on, if no other audio present at turn-off time

## Resource Reservation

- Video bandwidth requirements vary significantly
  - » low (200-400 Kb/s), medium (2-6 Mb/s), HDTV (15-20 Mb/s)
  - » backbone metalinks can support hundreds of channels
  - » audio bandwidth much less (32-200 Kb/s)
- Knowing number and quality of video channels, can estimate session bandwidth requirements
  - » reserve required bandwidth as branches added to tree
- When choosing paths for new endpoints, limit search to links with adequate unused bandwidth
- Also need processing resources for audio and video
  - » video source processing
    - to create preview – if 50 instructions/input byte, need 12.5 MIPS of processing power for each 2 Mb/s video stream
  - » video sink processing
    - create composite view with 2 main windows plus 4 previews
      - if 100 instructions/input byte, need about 60 MIPS
  - » if 3 way audio merge uses 30 inst/byte, need <1 MIPS for audio

# A Bulk Data Transfer Metanet



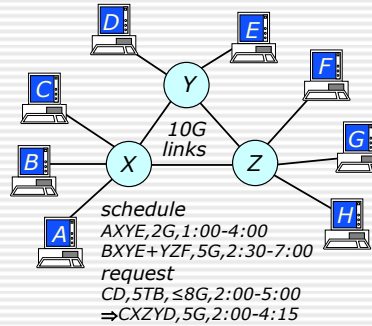
- Users request bulk data transfers in advance
  - » intended for 10 GB to 100 TB transfers
  - » specify availability time and target delivery time
  - » network schedules *pickup*, *transfer* and *delivery*
  - » scheduled reservation ensures contention-free transfer
- May add temporary metalinks for large transfers

# High Level Issues

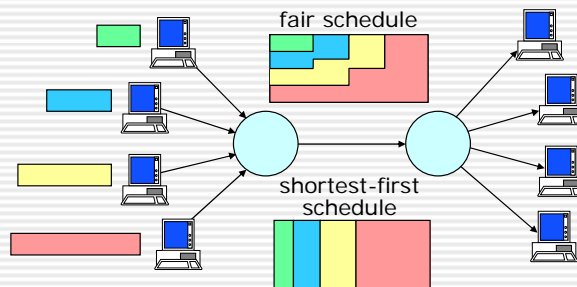
- Service models
  - » scheduled data transfers
    - network accepts/rejects transfer when user requests
    - network notifies user of scheduled window  $T$  time units before start
    - may specify periodic transfers and multi-destination transfers
  - » on-demand transfers – deliver ASAP
    - network regulates sending rate during transfer
- Resolve contention for resources in “fair” way
  - » prevent “hogging” of resources
    - limit reserved resources for any user – looser limit for near-term
  - » allocate resources based on users, not flows
    - requires user authentication, association of user with transfer
- Schedule maintenance
  - » simple approach
    - construct schedule incrementally, reject conflicting requests
  - » complex approach
    - continually restructure schedule as new requests come in
    - more computation, but higher throughput

# Scheduling Transfers

- Schedule is list of planned transfers with
  - » transfer topology
  - » bandwidth
  - » time window
- To handle request
  - » determine local schedules for links on possible paths
  - » use binary search to find fastest feasible rate
    - for specific rate & start time, easy to find path if one exists
    - find earliest feasible start time
- Performance requirements for advance scheduling
  - » for 100K users, scheduling average of 2 xfers/day
    - 200K transfers/day is average of 2 per second
    - estimate centralized server could handle 100-1000 per second
  - » for higher performance, use multiple servers
    - give each a “slice” of backbone, assign requests with DHT methods

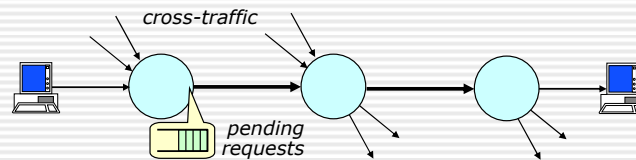


## Fairness for On-Demand Transfers



- Reconsidering fairness (from Gorinsky)
  - » traditional approach divides link equally among flows
  - » in shortest-first schedule, most finish earlier, none later
- More realistic cases
  - » for asynchronous arrivals, use smallest-remainder-first
  - » for bounded rate senders, allocate max allowed amount
  - » for general network topologies, need end-to-end rate allocation
    - requires rate allocation protocol and efficient allocation policies

## Rate Allocation for On-Demand Transfers



- Pending requests held in queue at backlogged link
- Before a transfer completes, pre-allocate its bandwidth
  - » to either pending request or another currently active transfer
  - » use smallest-remainder-first policy to order requests
  - » allocate bottleneck bandwidth on path to first request/transfer
    - iterate if more bandwidth available for allocation
  - » requires protocol to forward reservation request along path
    - tentative reservation, followed by confirmation (two round trips)
    - reserved bandwidth released within few seconds if no confirmation
- Path selected when initial request received
  - » select “short-enough path” with max available rate, min backlog