Introduction to Computer Networks

In-Network Support for TCP

CS640

https://pages.cs.wisc.edu/~mgliu/CS640/S26/index.html

Ming Liu
mgliu@cs.wisc.edu

Outline

- Last
  - TCP Congestion Control (III)
- Today
  - In-Network Support for TCP
- Announcements
  - Quiz 4 in-class next Thursday (04/23/2026)
  - Lab 4 due date 04/30/2026

Recap

- Key Questions:
  - What are the design goals of DCTCP?
  - How does DCTCP work?

- Terminology
  - ECN
  - Congestion extent

If we add some “intelligence” inside the network, can we improve the efficiency of the transport protocol?

#1: In-Network Resource Management

- Divide up resources among contending entities
  - Resources: network bandwidth and router/switch buffer space
  - Entities: transport flow, represented as five tuples

#1: In-Network Resource Management

- Divide up resources among contending entities
  - Resources: network bandwidth and router/switch buffer space
  - Entities: transport flow, represented as five tuples

- Congestion control is also an example of resource allocation
  - Run at the end-host
  - Run in a distributed manner without central coordination
  - Bandwidth = congestion window size / RTT

#1: In-Network Resource Management

Congestion control is a host-based, feedback-based distributed resource allocation scheme.

- Congestion control is also an example of resource allocation
  - Run at the end-host
  - Run in a distributed manner without central coordination
  - Bandwidth = congestion window size / RTT

#1: In-Network Resource Management

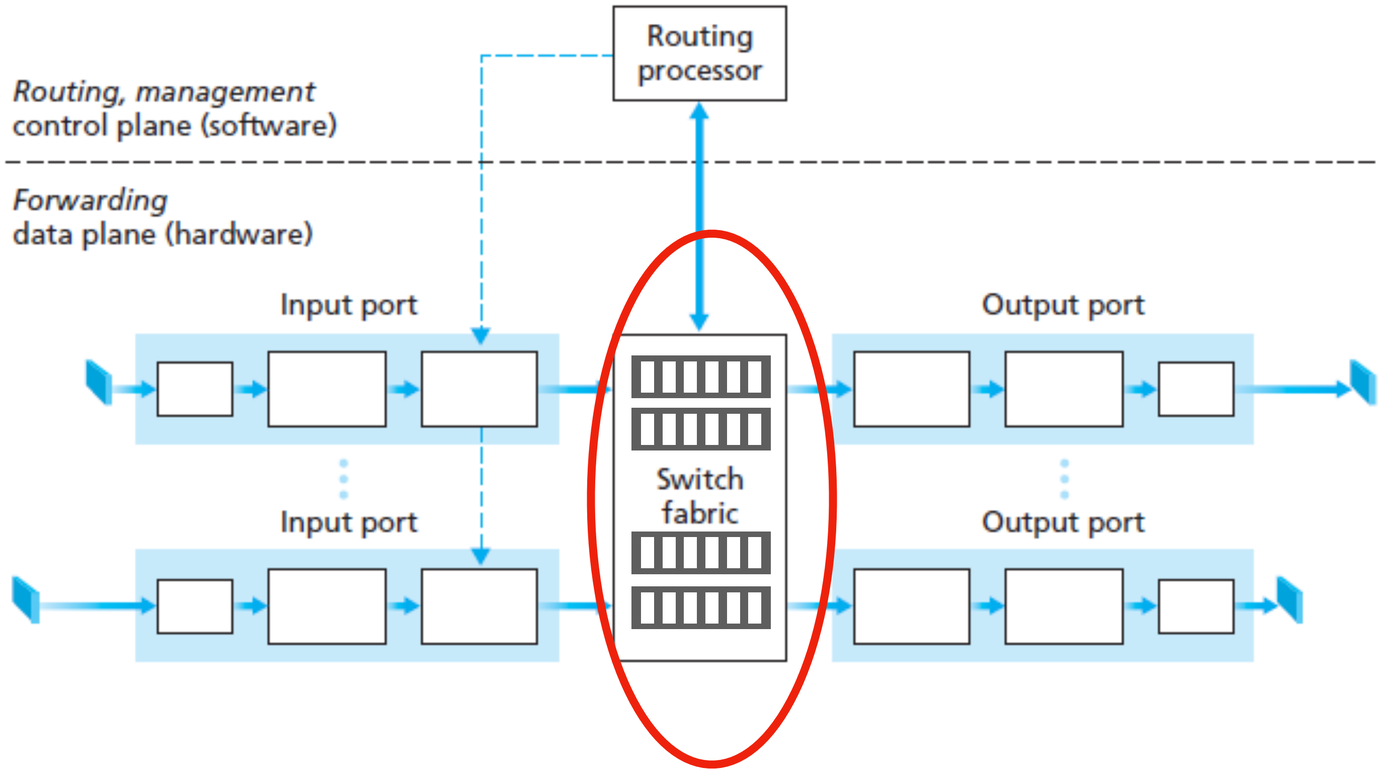
Congestion control is a host-based, feedback-based distributed resource allocation scheme.

- Congestion control is also an example of resource allocation
  - Run at the end-host
  - Run in a distributed manner without central coordination

Congestion control is a **router-assisted**, host-based, feedback-based distributed resource allocation scheme.

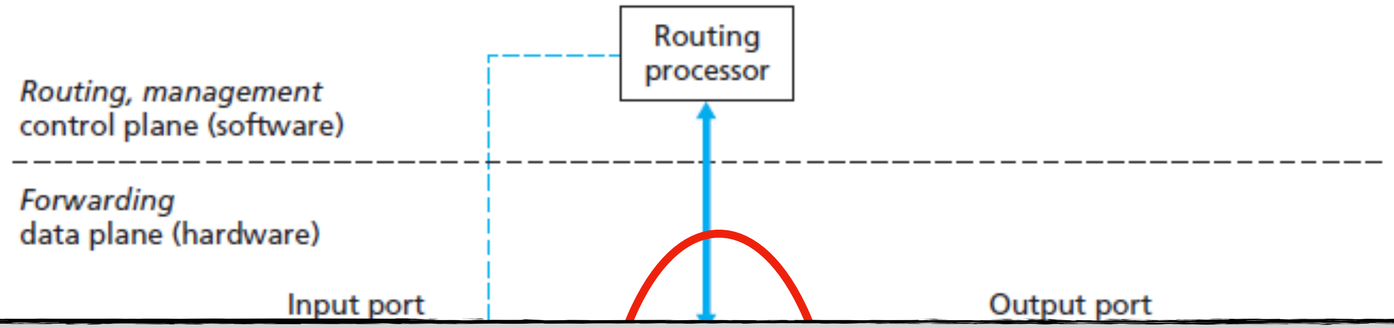
Active Queue Management (AQM)

- An “active” router/switch queue management
  - Facilitate better flow behavior under resource contention



Active Queue Management (AQM)

- An “active” router/switch queue management
  - Facilitate better flow behavior under resource contention



Two queueing disciplines

- Scheduling: which packets to determine on the dequeue side
- Dropping: which packets to drop on the enqueue side

Understanding Queueing

- Why queue build up?

Understanding Queueing

- Why queue build up?
  - Enqueuing rate > Dequeue rate

Understanding Queueing

- Why queue build up?
  - Enqueuing rate > Dequeue rate

- Why does queueing matter?

Understanding Queueing

- Why queue build up?
  - Enqueuing rate > Dequeue rate

- Why does queueing matter?
  - Bandwidth: which packet to serve (transmit) next
  - Buffer space: which packet to drop next (when required)

Understanding Queueing

- Why queue build up?
  - Enqueuing rate > Dequeue rate
- Why does queueing matter?
  - Bandwidth: which packet to serve (transmit) next
  - Buffer space: which packet to drop next (when required)
- Why does queueing impact performance?

Understanding Queueing

- Why queue build up?
  - Enqueuing rate > Dequeue rate
- Why does queueing matter?
  - Bandwidth: which packet to serve (transmit) next
  - Buffer space: which packet to drop next (when required)
- Why does queueing impact performance?
  - Queueing kicks in under contention
  - Scheduling a packet of flow i = allocate bandwidth for flow i
  - Dropping a packet from flow i = deallocate bandwidth for flow i

The Naive (But Widely-used) Approach

- Scheduling discipline: FIFO (first-in-first-out)
  - Dequeue packets based on the arrival order, not the flow priority

- Dropping discipline: Drop-Tail
  - Drop any packets when the queue is full, regardless of the flow priority

The issues

- #1: Lock-out problem
  - A few flows can easily monopolize the queue space
  - Lack of traffic isolation

- #2: Full queues
  - Make TCP adjust rates based on timeout
  - One might always observe bursty loss

In-network resource management: effectively use the router buffer under high network load

- Divide the buffer space equally among ongoing flows
- Notify the end hosts early to avoid bursty packet drops

In-network resource management: effectively use the router buffer under high network load

- Divide the buffer space equally among ongoing flows
- Notify the end hosts early to avoid bursty packet drops

Three techniques:

- #1: Fair Queueing (FQ)
- #2: Random Early Detection (RED)
- #3: Explicit Congestion Notification (ECN)

Technique #1: Fair Queueing

- Goal: allocate resources “fairly”
  - Keep an individual (virtual) queue for each flow

- Isolate ill-behaved users
  - The router does not send explicit feedback to the end-host
  - End-hosts still need an end-to-end congestion control

Max-Min Fairness

- A fair allocation scheme between demands and supplies
  - Allocate the user a “small” demand that it requires
  - Evenly distribute unused resources to “big” users

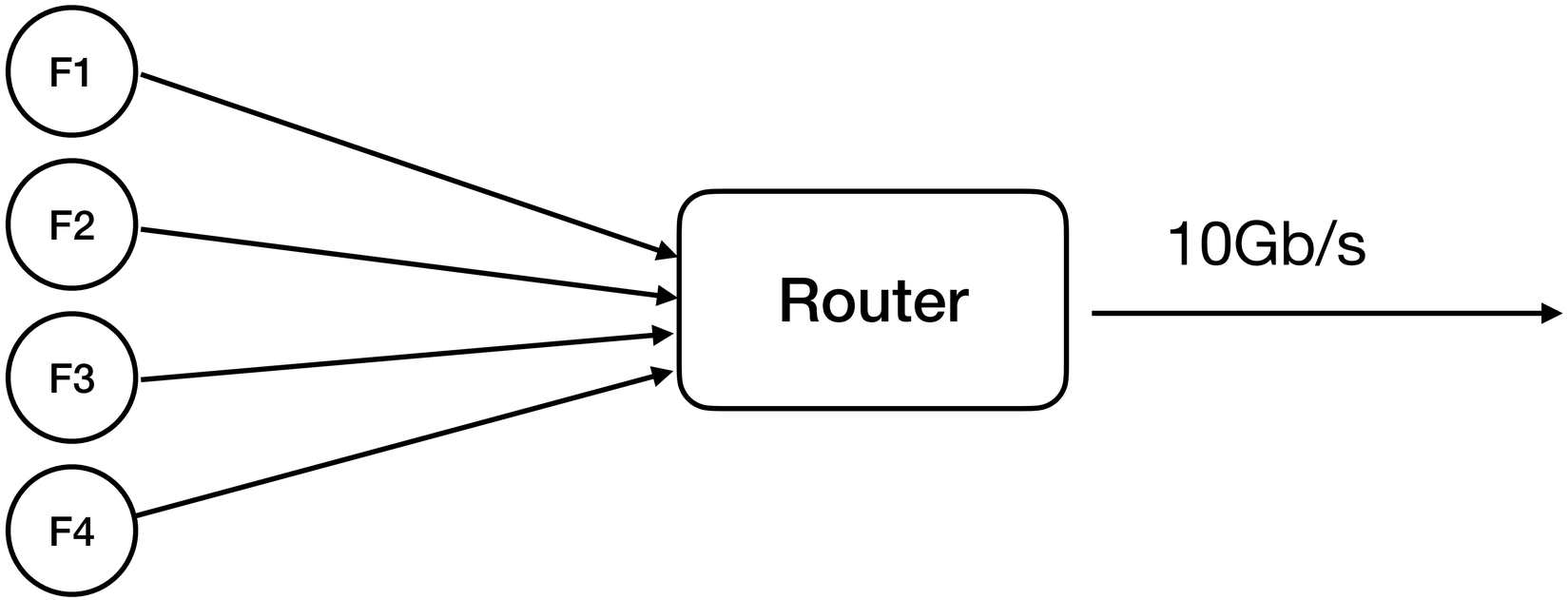
Max-Min Fairness

- A fair allocation scheme between demands and supplies
  - Allocate the user a “small” demand that it requires
  - Evenly distribute unused resources to “big” users

- Formally,
  - Resource allocated in terms of increasing demand
  - No sources get a resource share larger than its demands
  - Sources with unsatisfied demands get an equal share of the resources

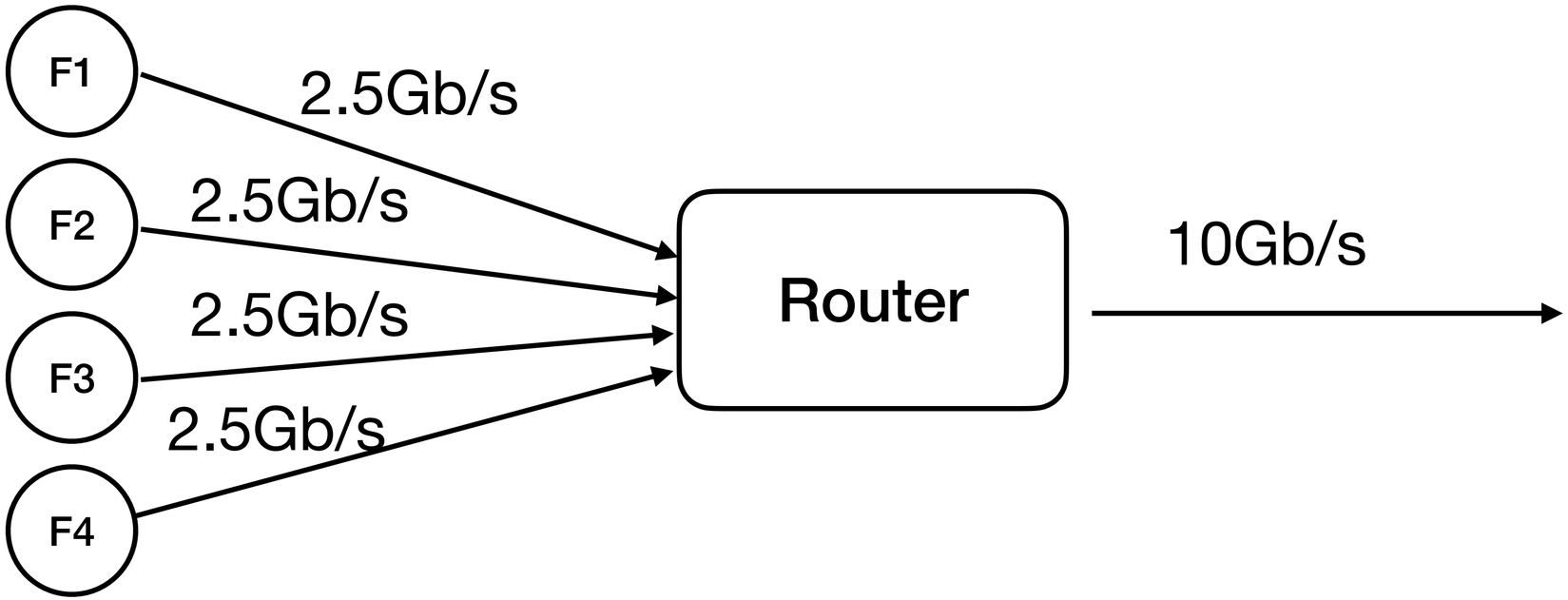
Max-Min Fairness Example

- Suppose four flows share 10Gb/s,
  - Considering fairness, how much bandwidth does each flow get?



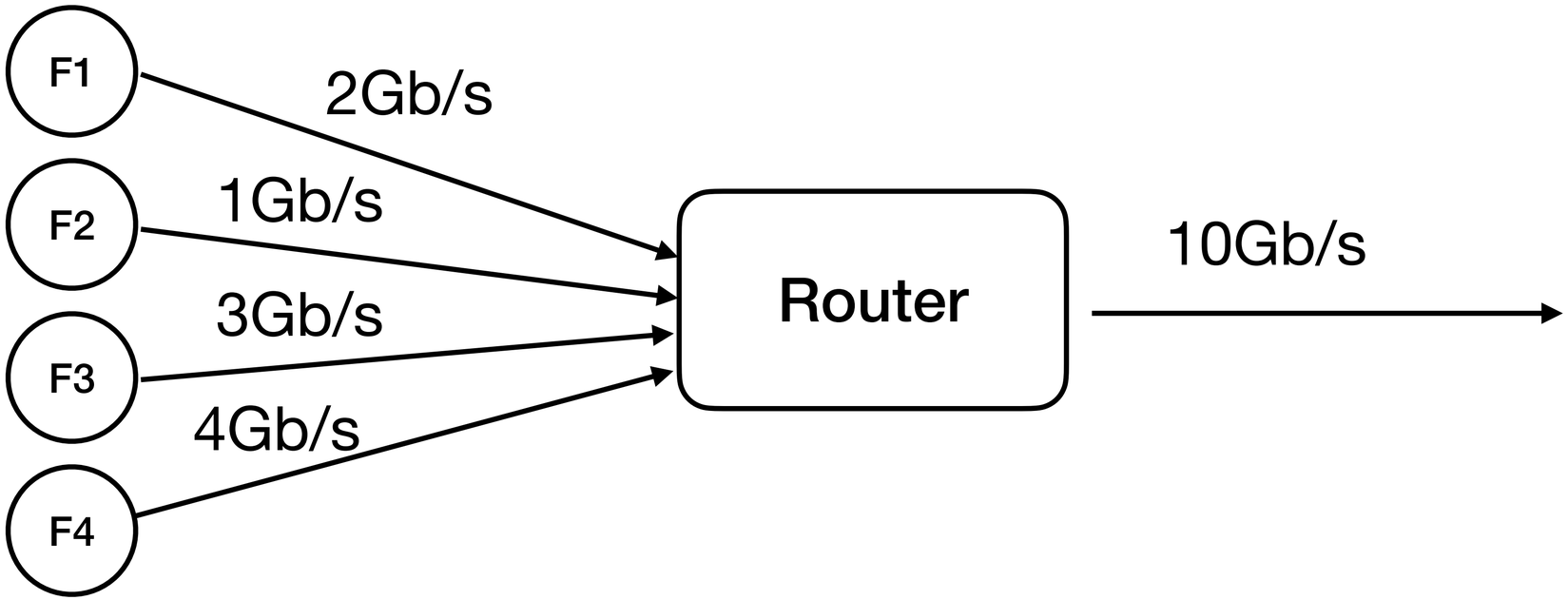
Max-Min Fairness Example

- Suppose four flows share 10Gb/s,
  - Considering fairness, how much bandwidth does each flow get?



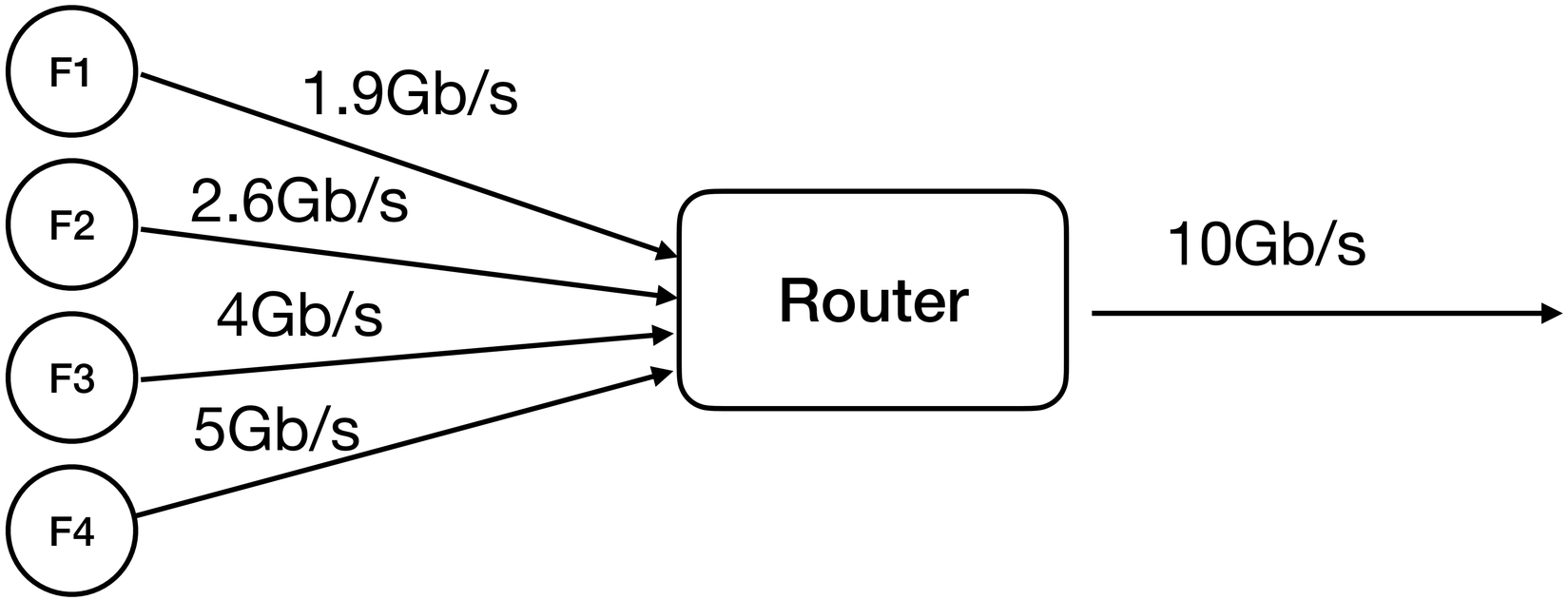
Max-Min Fairness Example

- Suppose four flows share 10Gb/s,
  - F1, F2, F3, F4 sends at 2Gb/s, 1Gb/s, 3Gb/s, 4Gb/s
  - Considering fairness, how much bandwidth does each flow get?



Max-Min Fairness Example

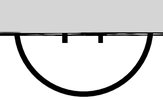
- Suppose four flows share 10Gb/s,
  - F1, F2, F3, F4 sends at 1.9Gb/s, 2.6Gb/s, 4Gb/s, 5Gb/s
  - Considering fairness, how much bandwidth does each flow get?



Max-Min Fairness Example

Round 1 (iteration 1):

- Each flow: 10Gb/s / 4 =2.5 Gb/s
- But F1 doesn't require 2.5 Gb/s
- Unused bandwidth is 0.6 Gb/s (from F1)



Max-Min Fairness Example

Round 1 (iteration 1):

- Each flow: 10Gb/s / 4 =2.5 Gb/s
- But F_1 doesn't require 2.5 Gb/s
- Unused bandwidth is 0.6 Gb/s (From F_1)

Round 2 (iteration 2):

- F_2, F_3, F_4: 0.6Gb/s / 3 =0.2 Gb/s
- Unused bandwidth is 0.1 Gb/s (From F_2)

Max-Min Fairness Example

Round 3 (iteration 3):

- F_3, F_4: 0.1Gb/s / 2 =0.05 Gb/s
- Unused bandwidth is 0 Gb/s

=> F_1: 1.9Gb/s, F_2: 2.6Gb/s, F_3: 2.75Gb/s, F_4: 2.75Gb/s

Implementing Max-Min Fairness

- Generalized processor sharing
  - Fluid fairness
  - Bitwise round-robin among all queues

- Why not a simple round-robin?
  - Variable packet length: Larger packets receive higher bandwidth
  - Unfair for instantaneous service rates
  - What if new packets arrive just before/after the packet departs?

Bit-by-bit Round Robin

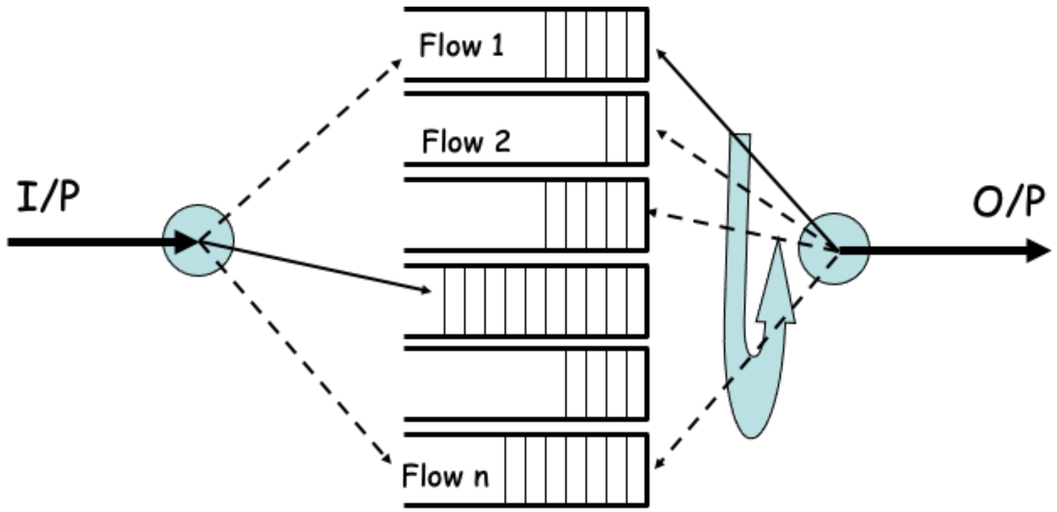
- Single flow: the clock ticks when a bit is transmitted. For packet i:
  - Pi = length (packet), Ai = arrive time
  - Si = begin transmit time, Fi = finish transmit time
  - Fi = Si + Pi = max (F(i-1), Ai) + Pi

- Multiple flows: clock ticks when a bit from all active flows is transmitted, defined as the round number
  - Can calculate Fi for a packet if the number of flows is known at all times

Fair Queueing Mechansim

- Mapping the bit-by-bit round-robin schedule onto transmission

- Transmit packet with the lowest Fi at any given time



Technique #2: Random Early Detection (RED)

- Key idea: detect incipient congestion

- Assume hosts respond to lost packets
  - Compliant congestion control

RED Algorithm

- Maintain a running average of queue length

- Case #1: if avg <= min_threshold, do nothing
  - Low queueing, send packets through
- Case #2: if avg >= max_threshold, drop packet
  - Protect from misbehaving sources

- Case #3: if min_threshold< avg < max_threshold, randomly drop
  - Calculate the probability P and drop the arriving packet with P
  - Notify the source that congestion is going to happen

RED Discussion

- Compute the average queue length (EWMA)
  - AvgLen = (1-weight) * AvgLen + weight * SampleLen, 0< weight <1
  - SampleLen is the queue length each time a packet arrives

- Probability P calculation
  - TempP = MaxP * (AvgLen - min_threshold) / (max_threshold - min_threshold)
  - P = TempP / (1-count * tempP)
  - Count = the number of newly arriving packets, while AvgLen stays between two thresholds (P increases with the count)

RED Discussion

- Compute the average queue length (EWMA)
  - AvgLen = (1-weight) * AvgLen + weight * SampleLen, 0< weight <1
  - SampleLen is the queue length each time a packet arrives

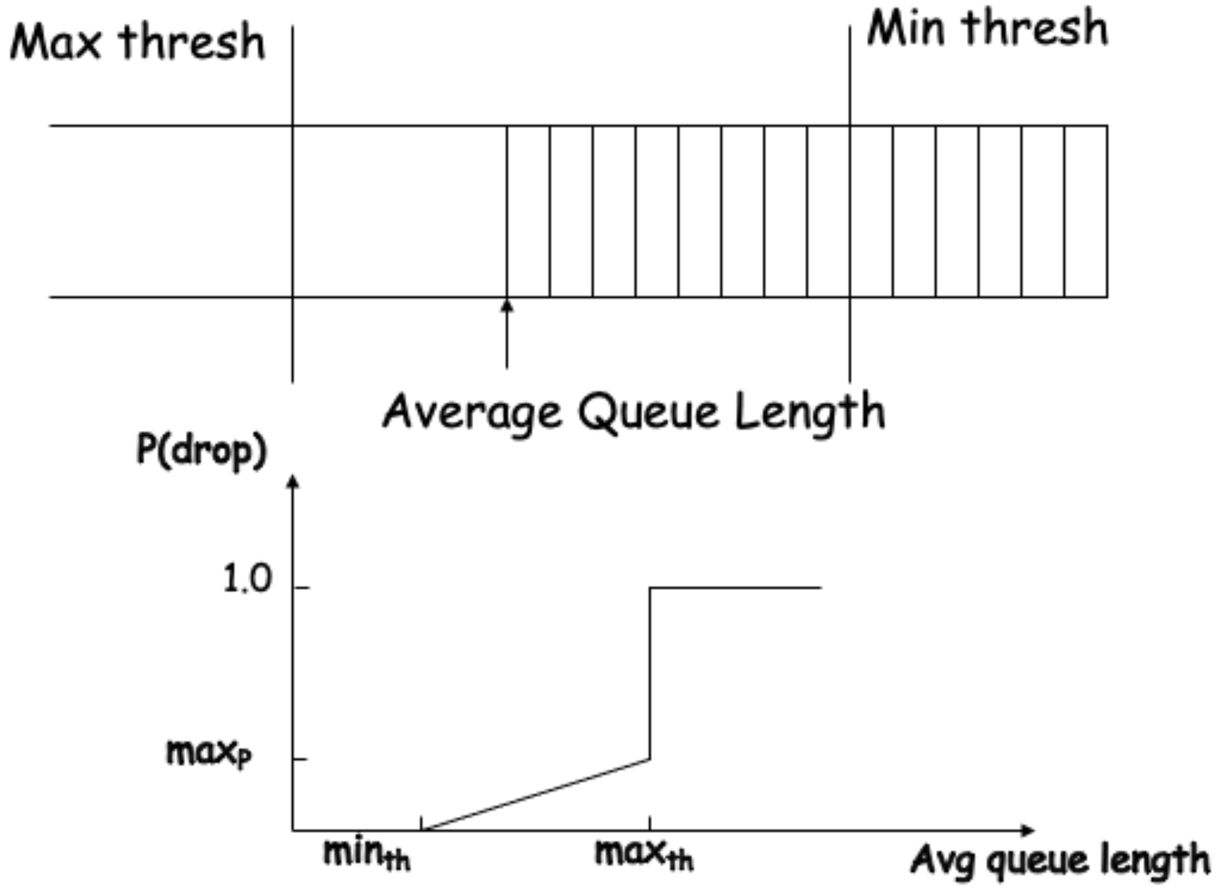
- Probability P calculation

Drop probability is a function of both AvgLen and how long it has been since the last drop

- TempP tracks how many newly arriving packets have been queued while AvgLen is between thresholds
- Count is the number of packets since the last drop
- This prevents clusters of drops

RED Operation

- RED is good at keeping average queue length steady

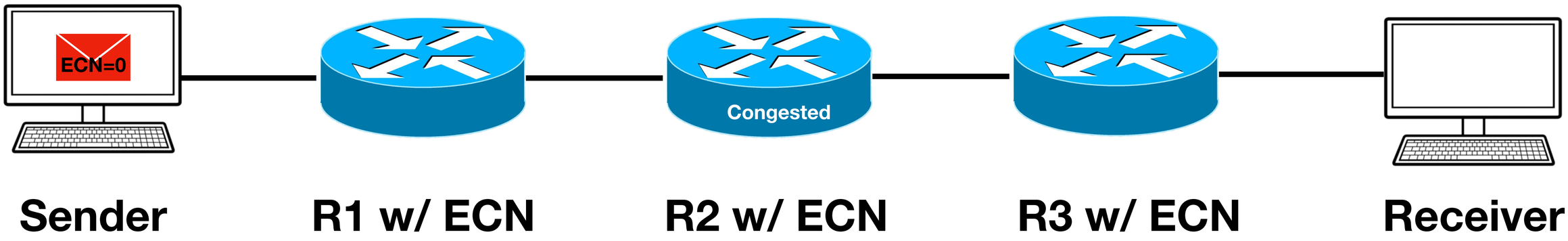


Technique #3: Explicit Congestion Notification (ECN)

- ECN allows end-to-end notification of network congestion
  - Sending path: An ECN-aware router may set a marker in the IP header instead of dropping a packet in order to signal impending congestion when the queue gets full
  - Receiving path: The receiver echos the congestion indication to the sender so that it can reduce the transmission rate

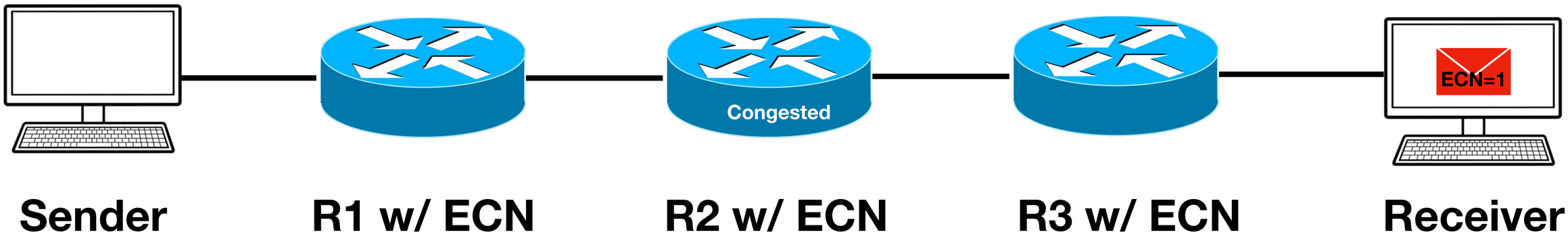
Technique #3: Explicit Congestion Notification (ECN)

- ECN allows end-to-end notification of network congestion
  - Sending path: An ECN-aware router may set a marker in the IP header instead of dropping a packet in order to signal impending congestion when the queue gets full
  - Receiving path: The receiver echos the congestion indication to the sender so that it can reduce the transmission rate



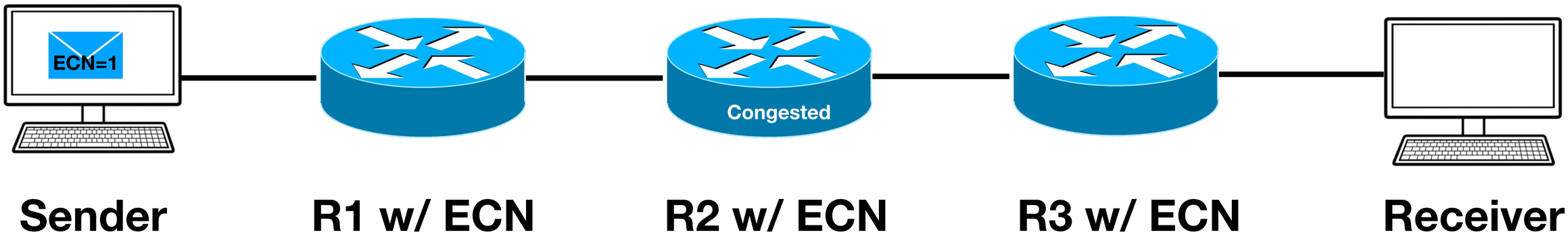
Technique #3: Explicit Congestion Notification (ECN)

- ECN allows end-to-end notification of network congestion
  - Sending path: An ECN-aware router may set a marker in the IP header instead of dropping a packet in order to signal impending congestion when the queue gets full
  - Receiving path: The receiver echos the congestion indication to the sender so that it can reduce the transmission rate



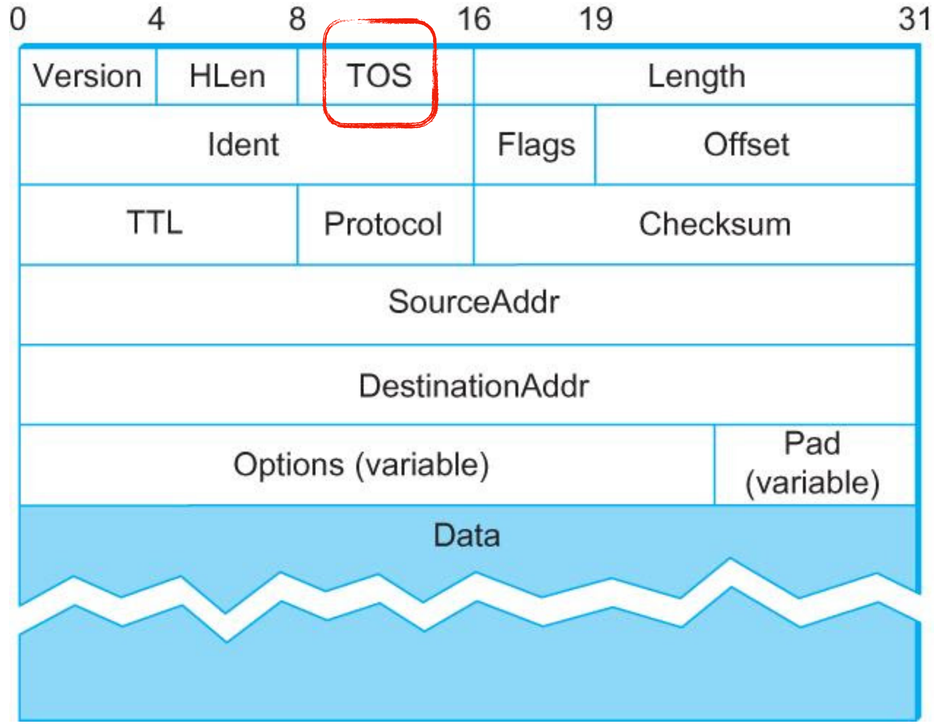
Technique #3: Explicit Congestion Notification (ECN)

- ECN allows end-to-end notification of network congestion
  - Sending path: An ECN-aware router may set a marker in the IP header instead of dropping a packet in order to signal impending congestion when the queue gets full
  - Receiving path: The receiver echos the congestion indication to the sender so that it can reduce the transmission rate



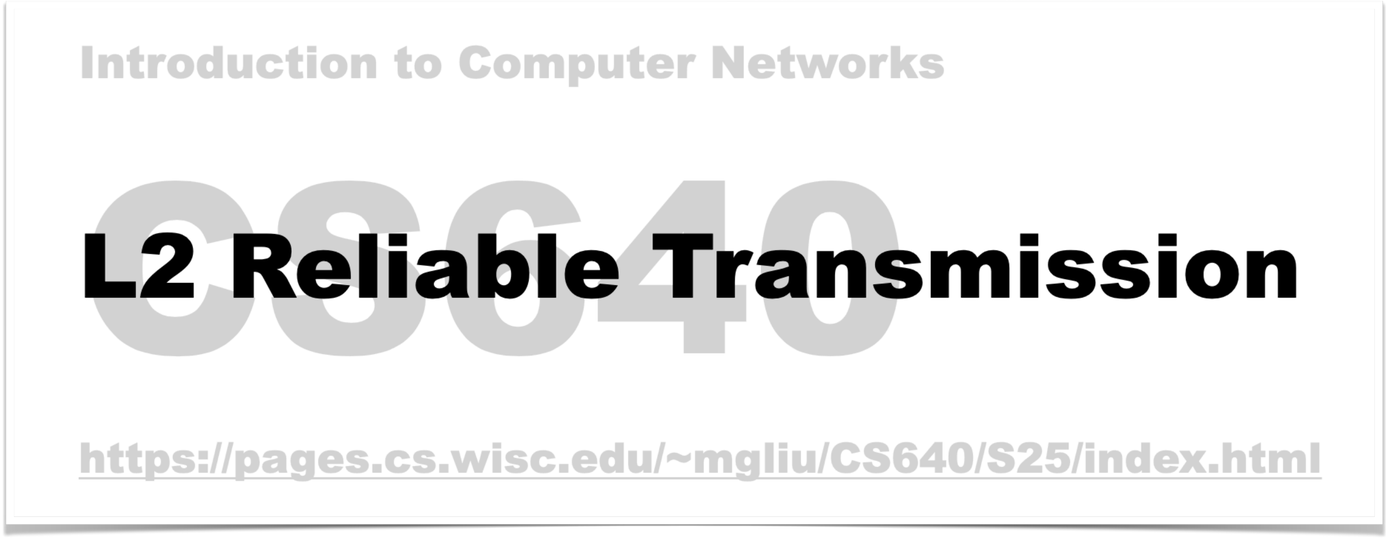
ECN in IP

- The two least significant bits in the traffic class field
  - 00 —> Non ECN-Capable Transport
  - 10/01 —> ECN Capable Transport
  - 11 —> Congestion Encountered



#2: Reliable Link Layer

- Packets are successfully delivered to the receiver
  - No packet drops
  - No packet duplications
  - No out-of-order delivery



#2: Reliable Link Layer

- Packets are successfully delivered to the receiver
  - No packet drops
  - No packet duplications
  - No out-of-order delivery

Introduction to Computer Networks

What does this mean for the congestion control?

https://pages.cs.wisc.edu/~mgliu/CS640/S25/index.html

TCP Congestion Control under Reliable Link Layer

- Can we see packet drops?

TCP Congestion Control under Reliable Link Layer

- Can we see packet drops?
  - Yes, but rarely happens, due to bit flips, etc.
  - But this does not indicate a network congestion

TCP Congestion Control under Reliable Link Layer

- Can we see packet drops?
  - Yes, but rarely happens, due to bit flips, etc.
  - But this does not indicate a network congestion
- Can we see packet duplication?

TCP Congestion Control under Reliable Link Layer

- Can we see packet drops?
  - Yes, but rarely happens, due to bit flips, etc.
  - But this does not indicate a network congestion
- Can we see packet duplication?
  - Yes, because the sender still needs to retransmit under failures

TCP Congestion Control under Reliable Link Layer

- Can we see packet drops?
  - Yes, but rarely happens, due to bit flips, etc.
  - But this does not indicate a network congestion
- Can we see packet duplication?
  - Yes, because the sender still needs to retransmit under failures
- Can we see out-of-order delivery?

TCP Congestion Control under Reliable Link Layer

- Can we see packet drops?
  - Yes, but rarely happens, due to bit flips, etc.
  - But this does not indicate a network congestion
- Can we see packet duplication?
  - Yes, because the sender still needs to retransmit under failures

So, nothing changes?

Congestion Control Enhancement

- #1: Implicit feedback —> Explicit feedback
  - No ACKs or NACKs indicate packets are dropped, so retransmitting
- #2: Explicit per-packet RTT
  - The value of RTT tells you the network condition
- #3: The following are still required, but not triggered frequently
  - Bandwidth probing
  - Flow control
  - Window adjustment

Congestion Control Enhancement

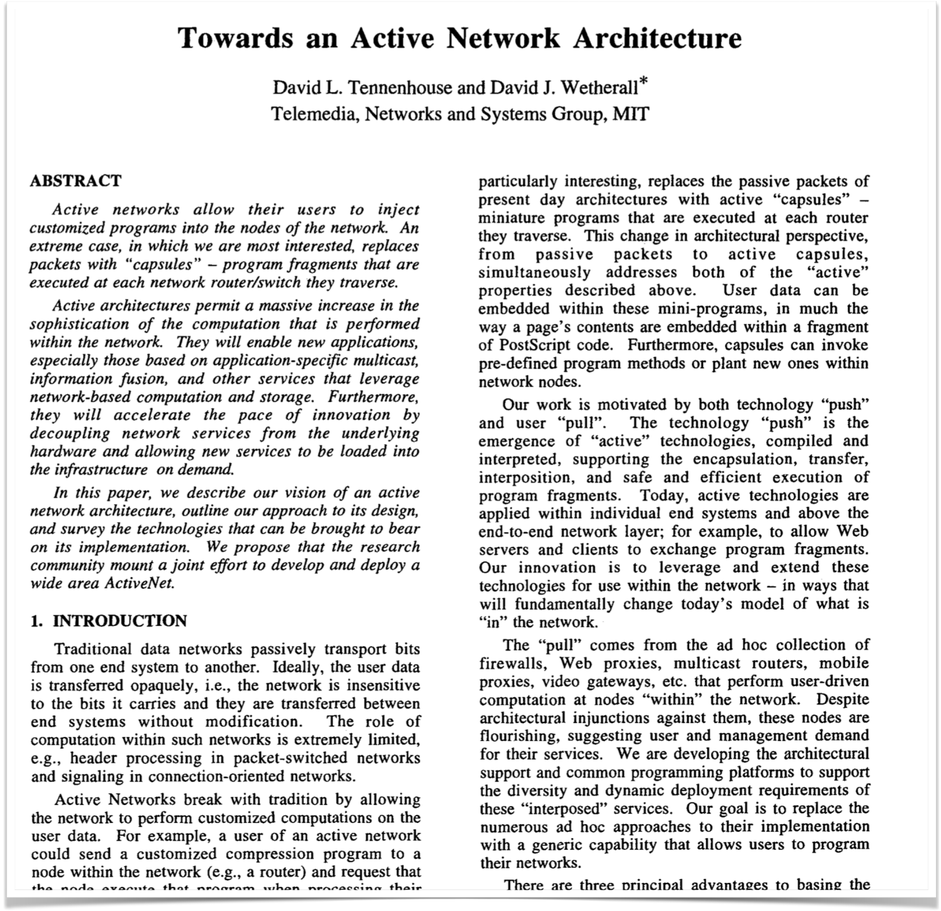
- #1: Implicit feedback —> Explicit feedback
  - No ACKs or NACKs indicate packets are dropped, so retransmitting

- #2: Explicit per-packet RTT
  - The value of RTT tells you the network condition

Your processor spends more cycles on application logics instead of doing TCP processing.
=> high communication and application performance

#3: Active Network

- A grant proposal that allows packets to carry programs
  - On-path entities can execute the logic



Towards an Active Network Architecture

David L. Tennenhouse and David J. Wetherall*
Telemedia, Networks and Systems Group, MIT

ABSTRACT

Active networks allow their users to inject customized programs into the nodes of the network. An extreme case, in which we are most interested, replaces packets with "capsules" – program fragments that are executed at each network router/switch they traverse.

Active architectures permit a massive increase in the sophistication of the computation that is performed within the network. They will enable new applications, especially those based on application-specific multicast, information fusion, and other services that leverage network-based computation and storage. Furthermore, they will accelerate the pace of innovation by decoupling network services from the underlying hardware and allowing new services to be loaded into the infrastructure on demand.

In this paper, we describe our vision of an active network architecture, outline our approach to its design, and survey the technologies that can be brought to bear on its implementation. We propose that the research community mount a joint effort to develop and deploy a wide area ActiveNet.

1. INTRODUCTION

Traditional data networks passively transport bits from one end system to another. Ideally, the user data is transferred opaquely, i.e., the network is insensitive to the bits it carries and they are transferred between end systems without modification. The role of computation within such networks is extremely limited, e.g., header processing in packet-switched networks and signaling in connection-oriented networks.

Active Networks break with tradition by allowing the network to perform customized computations on the user data. For example, a user of an active network could send a customized compression program to a node within the network (e.g., a router) and request that the node execute that program when processing their particularly interesting, replaces the passive packets of present day architectures with active "capsules" – miniature programs that are executed at each router they traverse. This change in architectural perspective, from passive packets to active capsules, simultaneously addresses both of the "active" properties described above. User data can be embedded within these mini-programs, in much the way a page's contents are embedded within a fragment of PostScript code. Furthermore, capsules can invoke pre-defined program methods or plant new ones within network nodes.

Our work is motivated by both technology "push" and user "pull". The technology "push" is the emergence of "active" technologies, compiled and interpreted, supporting the encapsulation, transfer, interposition, and safe and efficient execution of program fragments. Today, active technologies are applied within individual end systems and above the end-to-end network layer; for example, to allow Web servers and clients to exchange program fragments. Our innovation is to leverage and extend these technologies for use within the network – in ways that will fundamentally change today's model of what is "in" the network.

The "pull" comes from the ad hoc collection of firewalls, Web proxies, multicast routers, mobile proxies, video gateways, etc. that perform user-driven computation at nodes "within" the network. Despite architectural injunctions against them, these nodes are flourishing, suggesting user and management demand for their services. We are developing the architectural support and common programming platforms to support the diversity and dynamic deployment requirements of these "interposed" services. Our goal is to replace the numerous ad hoc approaches to their implementation with a generic capability that allows users to program their networks.

There are three principal advantages to basing the

Programmable Networks

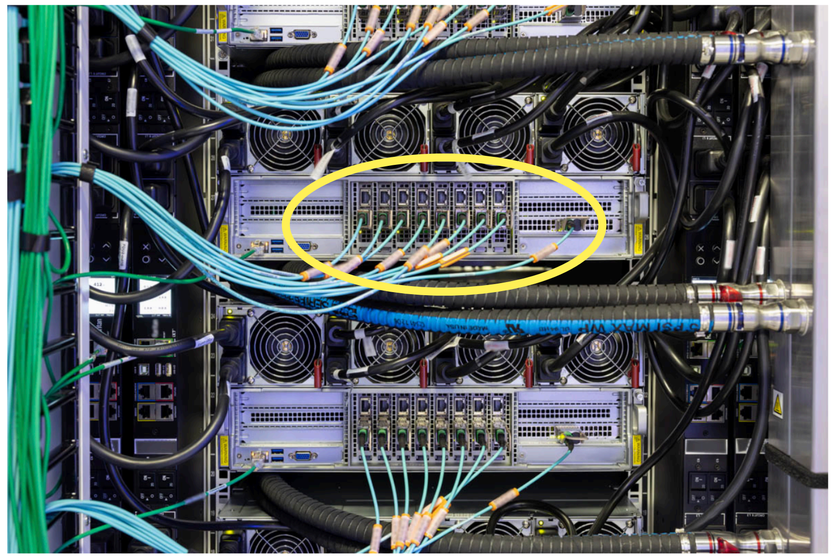
- Programmable switch
  - Reconfigurable match-action tables, registers, and TCAMs
  - E.g., Cavium/Marvell (ceased in 2018), Barefoot/Intel (ceased in 2023), Mellanox/NVIDIA Spectrum, and Juniper Trio

- SmartNIC
  - NICs with general-purpose processors, domain-specific accelerators, programmable DMA engines, and SRAM/DRAM
  - E.g., AWS Nitro solution, Pensando/AMD DSC, NVIDIA BlueField (heavily deployed in the xAI Colossus cluster), Intel/Google Mount Evans, Microsoft FPGA (and/or Fungible DPU)

SmartNICs in the Real World

- Case #1: AWS nitro card
  - Annapurna Labs acquired in 2015
  - Packet processing and I/O data-plane offloading

- Case #2: Colossus cluster from xAI
  - 100K NVIDIA H100 GPUs
  - 400GbE Ethernet
  - BlueField-3 SuperNIC + Spectrum-X
  - One SmartNIC One GPU



Summary

- Today
  - TCP in-network support

- Next lecture
  - Linux NStack and Infrastructure Services