

# Computer Networks

*Prof.(Dr.) Yuh-Jong Hu*

*2005/02/17 - 2005/06/23*

*hu@cs.nccu.edu.tw*

*<http://www.cs.nccu.edu.tw/~jong>*

*Emerging Network Technology(ENT) Lab.*

*Department of Computer Science*

*National Chengchi University, Taipei, Taiwan*

## *Course Syllabus*

計 算 機 網 路  
(Computer Networks)

2005 年春季班

指導老師：胡毓忠 教授

上課時間： 星期四 早上 9：10 至 12：00

上課地點： 大仁樓 3 樓 200102

老師辦公室：大仁樓 3 樓 200313

電話：2938-7620

電子郵件：jong@cs.nccu.edu.tw

網址：<http://www.cs.nccu.edu.tw/~jong>

問問題時間：星期一和二早上 10：00 至中午 12：00

課程助教： 余承遠 (g9302@cs.nccu.edu.tw)

教科書：Computer Networks

A System Approach, 3<sup>rd</sup> Edition

作者：Larry L. Peterson & Bruce S. Davie

出版社：Morgan Kaufmann

進口書商：新月圖書公司 電話：2331-7856, 2331-1578

參考書：Computer Networking with Internet Protocols and Technology

作者： William Stallings

出版社： Pearson, Prentice Hall, 2004

進口書商：高立圖書有限公司 電話：(02)2290-0318 轉 271

Java Network Programming

作者： Merlin Hughes et al.

出版社：Prentice Hall, 1999

Computer Networks and Internets

作者：Douglas E. Comer

出版社： Prentice Hall, 1997

TCP/IP Illustrated, Volume 1 The Protocols

作者： W. Richard Stevens

出版社：Addison-Wesley, 1994

TCP/IP Illustrated, Volume 2 The Implementation

作者：Gary R. Wright, W. Richard Stevens

出版社：Addison-Wesley, 1995

#### 上課進度及上課大綱

日期	內容
2/17	Background Introduction
2/24	Introduction <ul style="list-style-type: none"><li>1.1 Applications</li><li>1.2 Requirements</li><li>1.3 Network Architecture</li><li>2.1 Hardware Building Blocks</li></ul>
3/3	Implementation Issues <ul style="list-style-type: none"><li>1.4 Implementing Network Software</li><li>1.5 Performance</li></ul>
3/10	Point-to-Point Links <ul style="list-style-type: none"><li>2.2 Encoding</li><li>2.3 Framing</li><li>2.4 Error Detection</li><li>2.5 Reliable Transmission</li></ul>
3/17	Shared Media Networks <ul style="list-style-type: none"><li>2.6 Ethernet</li><li>2.7 Token Rings (802.5, FDDI)</li><li>2.8 Wireless (802.11)</li></ul>
3/24	Switched Networks (Packet Switching) <ul style="list-style-type: none"><li>3.1 Switching and Forwarding</li><li>3.2 Bridges and LAN Switches</li><li>3.3 Cell Switching (ATM)</li></ul>

3/31	IP and the Internet (Internetworking) 4.1 Simple Internetworking (IP)
4/7	春假放假
4/14	期中考試 (繳交群體計畫的設計書)
4/21	Scalable Routing 4.2 Routing 4.3 Global Internet
4/28	TCP and UDP (End-to-End Protocols) 5.1 Simple Demultiplexer (UDP) 5.2 Reliable Byte Stream (TCP)
5/5-5/12	Congestion Control 6.1 Issues in Resource Allocation 6.2 Queuing Disciplines 6.3 TCP Congestion Control 6.4 Congestion-Avoidance Mechanisms
5/19	Quality of Service 6.5 Quality of Service
5/26	Remote Procedure Call 5.3 Remote Procedure Call 7.1 Presentation Formatting
6/2	Naming and Security 9.1 Name Service (DNS) chap. 8: Network Security
6/9	Overlay Networks 9.4.1 Peer-to-Peer (P2P) Networks 9.4.2 Peer-to-Peer Networks 9.4.3 Content Distribution Networks

6/16

期末考試（群體計畫一星期之內驗收完畢）

6/23

群體計畫驗收截止日及學期個人研究報告繳交截止日

## 學期個人研究報告

本學期每位修課的同學都要提出一份 15 頁以內的研究報告，主要是探討網際網路語音 (Voice over IP, VoIP) 交換和對等式 (Peer-to-Peer, P2P) 資訊系統結合是否有可能取代電話網路系統的可行性研究，你的研究報告必須要有根據你自己的看法和相關文獻的資料得出「是或否」的具體結論。我們都知道最近一兩年來 VoIP 的技術快速的發展其未的趨勢有可能要和現有以線路交換為主的傳統電話網路來相融合或者相抗衡。早期傳統式 VoIP 技術主要是以公用電話網路 (Public Switched Telephone Network, PSTN) 為主體，使用者發話端可以使用一般電話網路並經過閘門 (gateway) 的轉換可以連上數據網路並以網際網路的 TCP/IP 協定用分封交換的方式來傳送語音封包並經過另一閘門的轉換將語音數據封包轉給使用者接收端的電話網路。傳統式的 VoIP 最大的問題因為無法有效控管語音封包的等候延遲時間，所以造成通話失真且效能不彰的情況。因此在能量上無法和既有的 PSTN 電話網路相抗衡。

新一代的 VoIP 如 Skype (<http://www.skype.com>) 則是以網際網路為通話的骨幹網路，使用者端不論發話者或者受話者將以電腦為主要的末端溝通工具，因此所有的電腦周邊設備可以有無限的擴展空間。而只要登錄上線的個人都可以透過系統本身相互連線並以語音的方式完成一對一或多對多溝通的目的。配合寬頻網路和對等式資訊系統技術的發展，溝通的效能及延遲等候時間都能夠大幅度的提昇。除此之外對於使用者登入登出及相關記錄檔的管理和查詢都可以彈性選擇將其落實到使用者端自身的環境或登錄伺服器主機之中，以避免隱私權受到侵犯。同樣的，只要透過適當的閘門，如 Skype 的 VoIP 網路也可以 Skype Out 到一般傳統式的 PSTN 電話網路以達成和現有語音網路使用者相通的目的。

在學期上課進行之中老師將會提供相關的文獻供各位參考和研讀以方便各位研究報告的繕寫和討論，論文的格式也將在上課之後一併規範。

## 學期群體系統開發計畫

本學期的群體計畫將要開發一個可以利用跨平台以即時簡訊 (Instant Message)進行資訊交換及共享的網路應用平台。相類似的系統可以參考微軟的 MSN Messenger 系統 (詳見 <http://www.msn.com.tw/>)。在這個即時簡訊系統中，使用者可以設定和他自身有關的各種通訊群組來進行即時的線上訊息交換和資料的共享。詳細的規範書(Specifications)會在開學後數週之內給予，本群體計畫將以三個人為一組的人數最高上限，請同學在第二次（下次）上課（2月24日）時給我分組的名單。同學必須遵循這個規範書設計一個系統架構和分工報告計畫書並在期中考試當天交出，並列入學期群體計畫的分數考量。我們將在本課程的期末考試完畢之後一星期之內完成整體系統的測試和驗收。

為了讓大家能夠快速的建立網路程式寫作的能力，我會請助教設計一些簡單的範例讓大家瞭解如何用 Java (or C) 的 Stream (or Socket)來進行網路上程式和程式的溝通。這些完整的範例可以參考本課程的參考書：Java Network Programming。這裡面提供了一些實際範例來告訴我們如何建構一個線上交談 (Chat)的系統，它和本學期的群體計畫即時簡訊的有些功能非常相類似。

## 學期成績評分方式及標準

學期成績的計算將以學期個人研究報告、學期群體系統開發計畫（和期中報告）、2次考試（期中及期末）、的各項分數加總之後為基礎來將全班的成績調整到平均 75 分左右。學生個別的成績如果經過調整之後仍不滿 60 分，將以不及格的方式處理沒有例外。另外基於公平的原則，每次作業繳交、考試、及上機測試如果無法在截止（預定）時間之內完成(含遲交)或者缺考者，則該次的分數將以零分計算，沒有補救（考）的機會。

### 評分標準

學期個人研究報告	15%
學期系統群體計畫（含期中報告）	25%
期中考試	30%
期末考試	30%

# Talk Outline

✧ *Background Introduction*

✧ *Introduction*

✧ *Implementation Issues*

✧ *Point-to-Point Links*

✧ *Shared Media Networks*

✧ *Switched Networks  
(Packet Switching)*

✧ *IP and the Internet  
(Internetworking)*



# Talk Outline (conti.)

- ✧ *Scalable Routing*
- ✧ *TCP and UDP*  
*(End-to-End Protocols)*
- ✧ *Congestion Control*
- ✧ *Quality of Service*
- ✧ *Remote Procedure Call*
- ✧ *Naming and Security*
- ✧ *Overlay Networks*

# *Background Introduction*

*Go To Talk Outline*

# *Introduction*

*Go To Talk Outline*

# Introduction

## Outline

- Statistical Multiplexing
- Inter-Process Communication
- Network Architecture
- Performance Metrics
- Implementation Issues

# Building Blocks

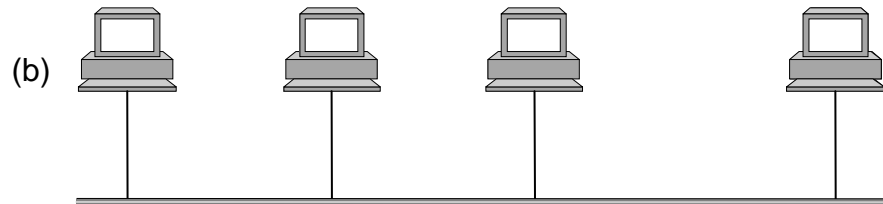
- Nodes: PC, special-purpose hardware...
  - hosts
  - switches

- Links: coax cable, optical fiber...

- point-to-point

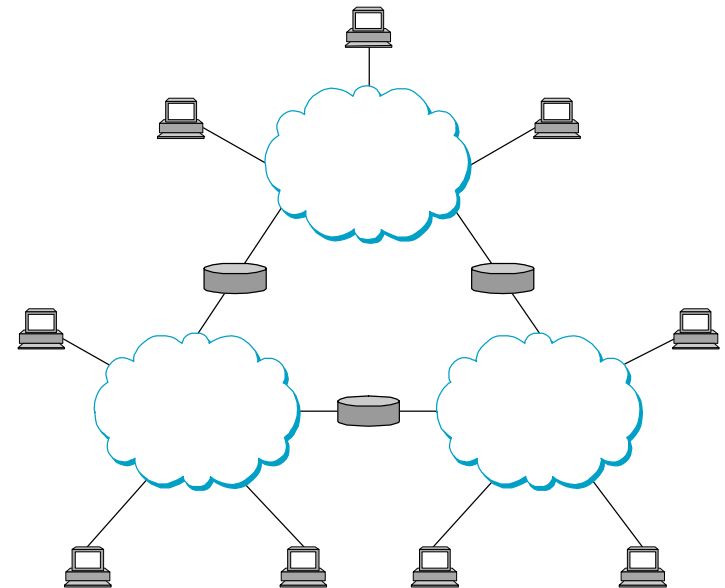
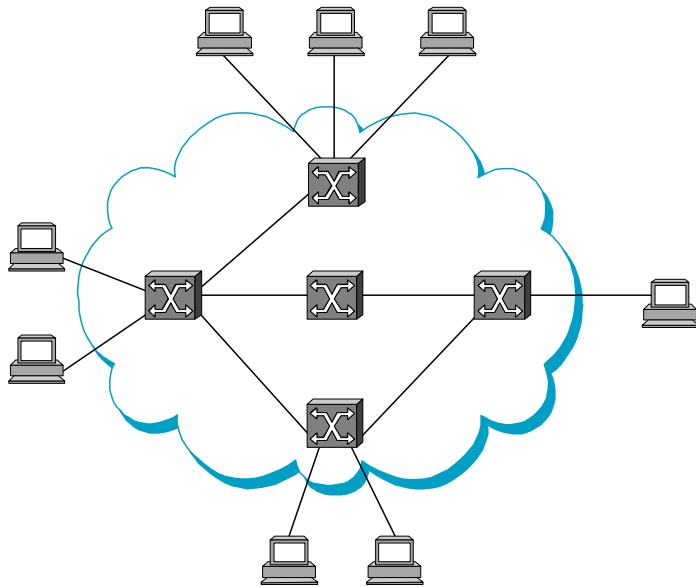


- multiple access



# Switched Networks

- A network can be defined recursively as...
  - two or more nodes connected by a link, or
  - two or more networks connected by a node



# Strategies

- Circuit switching: carry bit streams
  - original telephone network
- Packet switching: store-and-forward messages
  - Internet

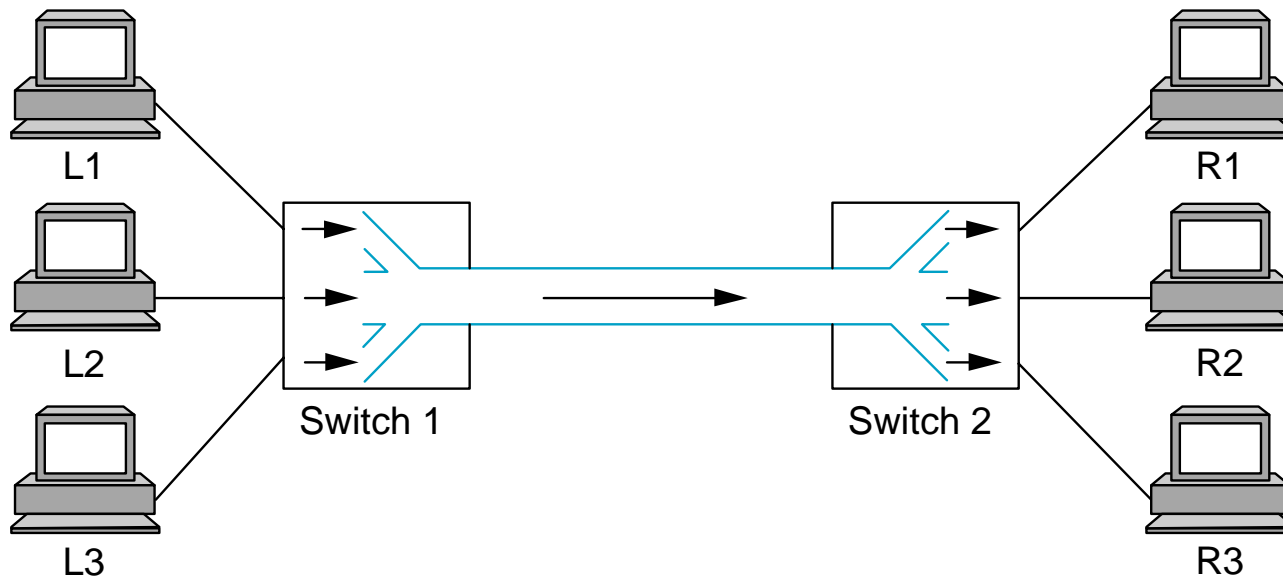
# Addressing and Routing

- Address: byte-string that identifies a node
  - usually unique
- Routing: process of forwarding messages to the destination node based on its address
- Types of addresses
  - unicast: node-specific
  - broadcast: all nodes on the network
  - multicast: some subset of nodes on the network



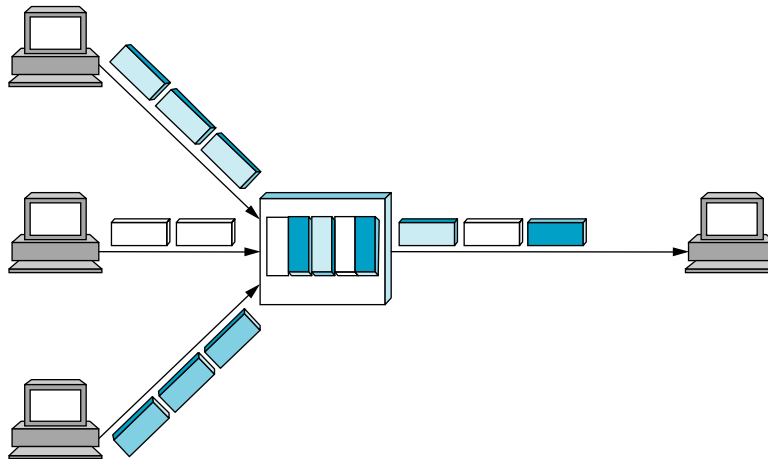
# Multiplexing

- Time-Division Multiplexing (TDM)
- Frequency-Division Multiplexing (FDM)



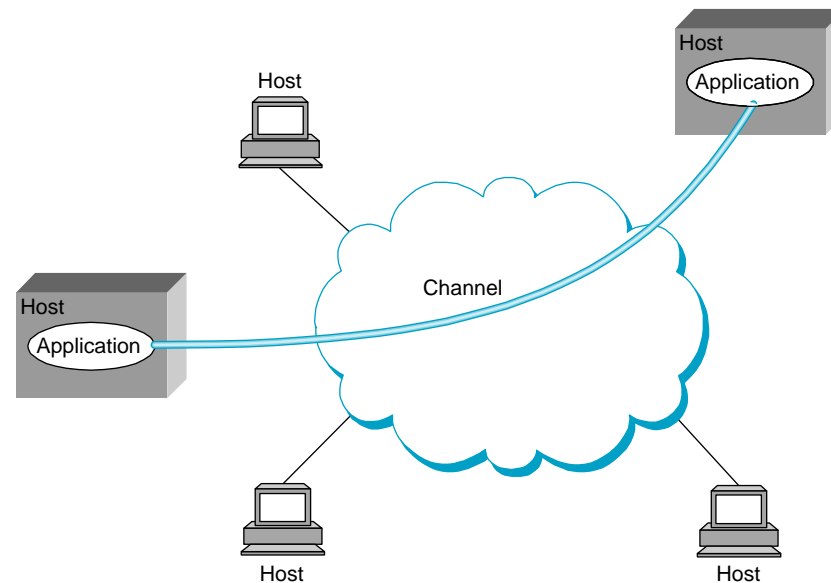
# Statistical Multiplexing

- On-demand time-division
- Schedule link on a *per-packet* basis
- Packets from different sources interleaved on link
- Buffer packets that are *contending* for the link
- Buffer (queue) overflow is called *congestion*



# Inter-Process Communication

- Turn host-to-host connectivity into process-to-process communication.
- Fill gap between what applications expect and what the underlying technology provides.



# IPC Abstractions

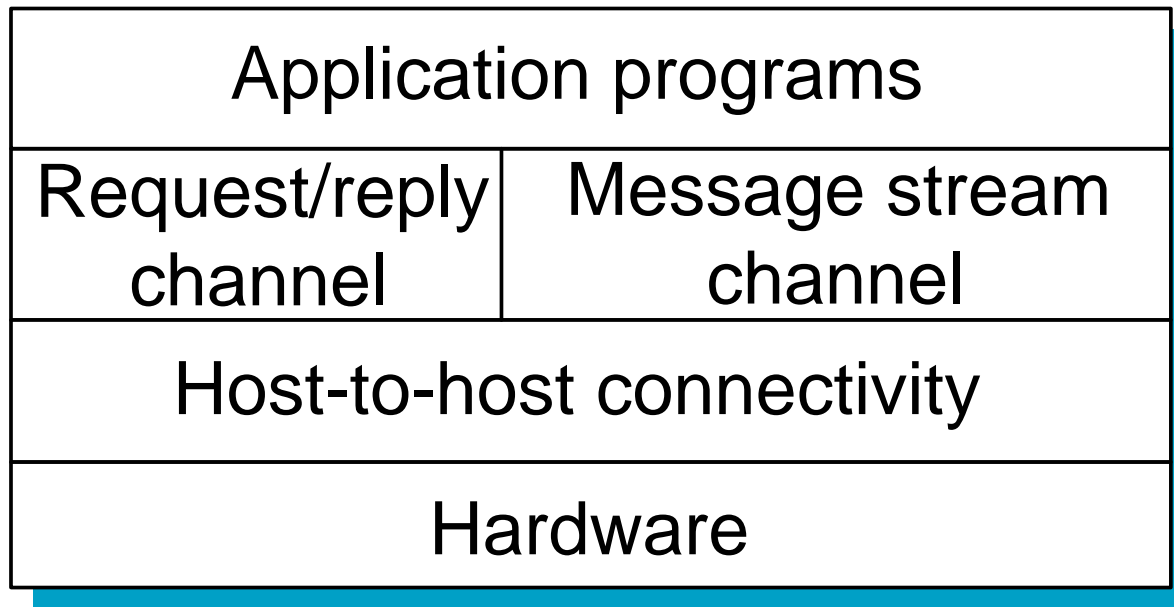
- Request/Reply
  - distributed file systems
  - digital libraries (web)
- Stream-Based
  - video: sequence of frames
    - 1/4 NTSC = 352x240 pixels
    - $(352 \times 240 \times 24)/8 = 247.5\text{KB}$
    - 30 fps = 7500KBps = 60Mbps
  - video applications
    - on-demand video
    - video conferencing

# What Goes Wrong in the Network?

- Bit-level errors (electrical interference)
- Packet-level errors (congestion)
- Link and node failures
  
- Packets are delayed
- Packets are deliver out-of-order
- Third parties eavesdrop

# Layering

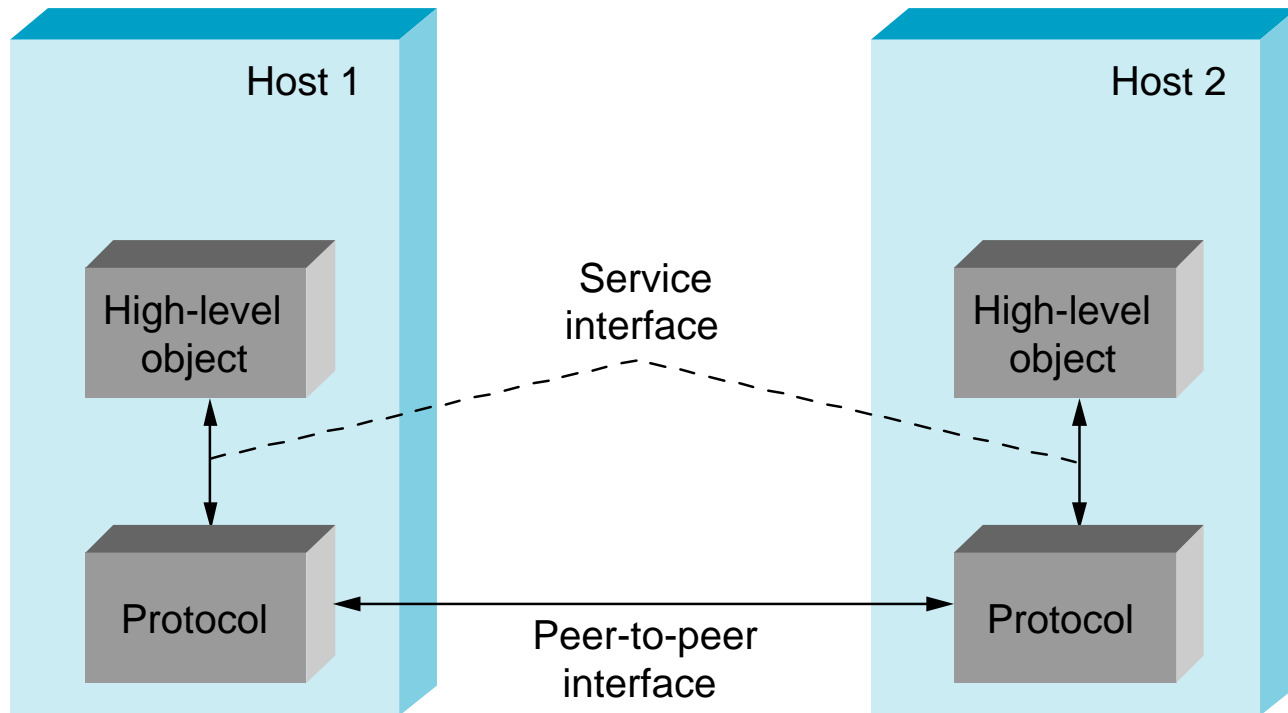
- Use abstractions to hide complexity
- Abstraction naturally lead to layering
- Alternative abstractions at each layer



# Protocols

- Building blocks of a network architecture
- Each protocol object has two different interfaces
  - *service interface*: operations on this protocol
  - *peer-to-peer interface*: messages exchanged with peer
- Term “protocol” is overloaded
  - specification of peer-to-peer interface
  - module that implements this interface

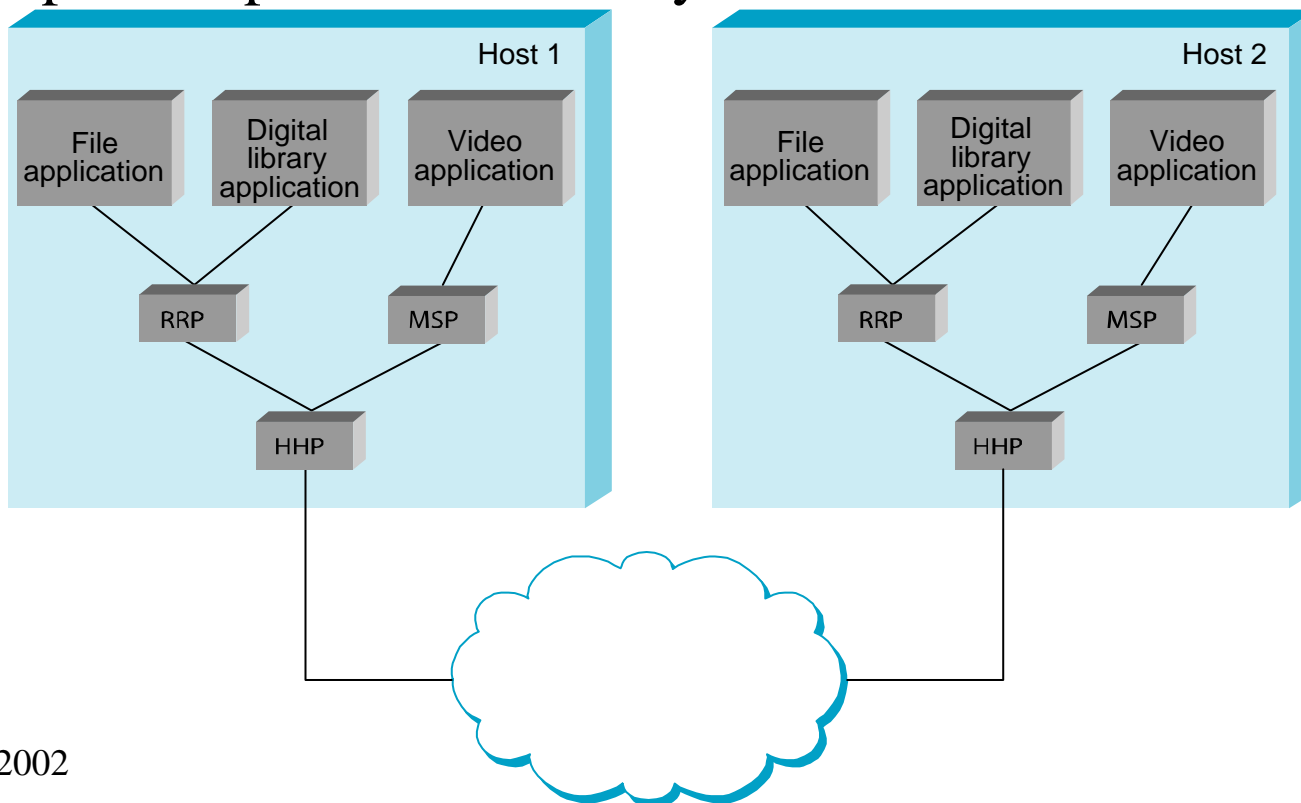
# Interfaces





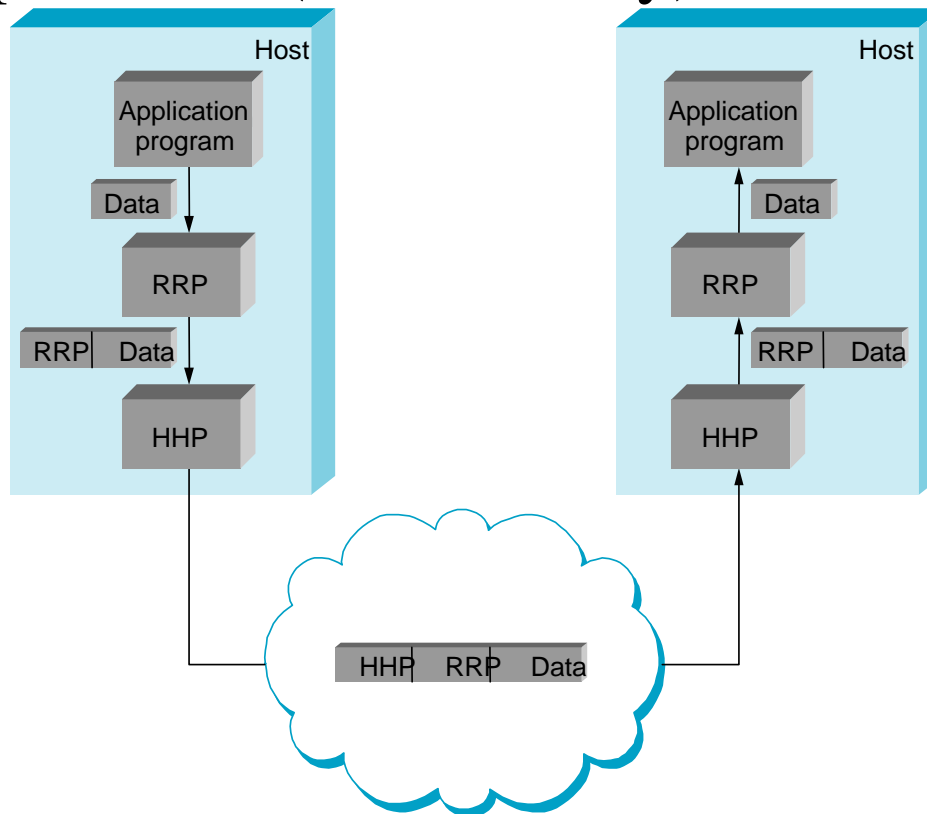
# Protocol Machinery

- Protocol Graph
  - most peer-to-peer communication is indirect
  - peer-to-peer is direct only at hardware level



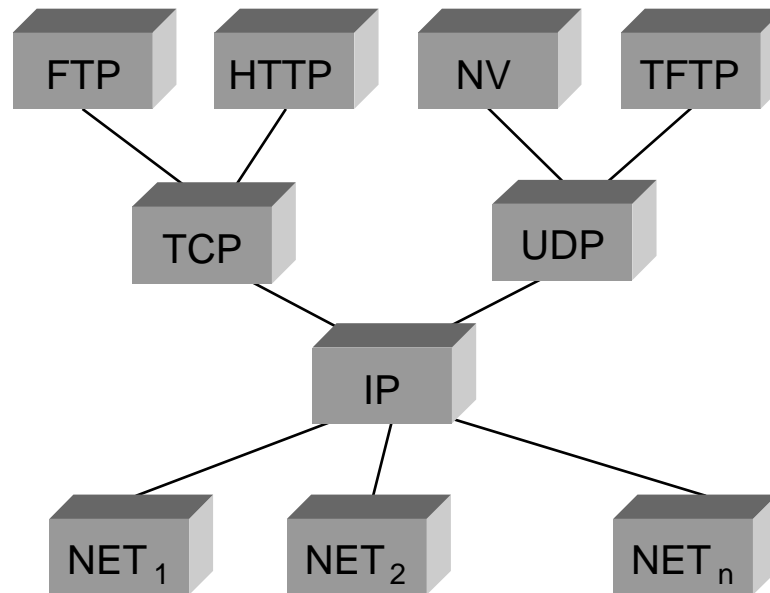
# Machinery (cont)

- Multiplexing and Demultiplexing (demux key)
- Encapsulation (header/body)

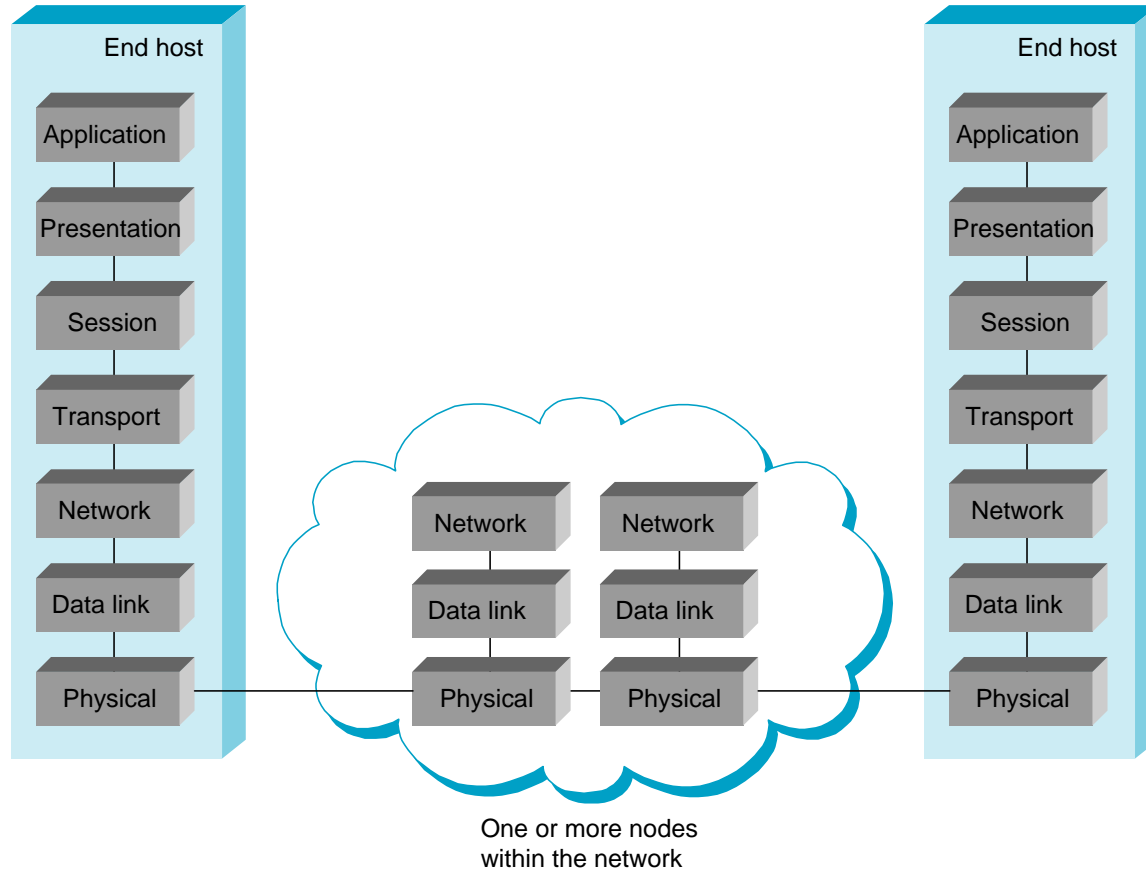


# Internet Architecture

- Defined by Internet Engineering Task Force (IETF)
- Hourglass Design
- Application vs Application Protocol (FTP, HTTP)



# ISO Architecture



# Performance Metrics

- **Bandwidth (throughput)**
  - data transmitted per time unit
  - link versus end-to-end
  - notation
    - $\text{KB} = 2^{10}$  bytes
    - $\text{Mbps} = 10^6$  bits per second
- **Latency (delay)**
  - time to send message from point A to point B
  - one-way versus round-trip time (RTT)
  - components
    - Latency = Propagation + Transmit + Queue
    - Propagation = Distance / c
    - Transmit = Size / Bandwidth

# Bandwidth versus Latency

- Relative importance
  - 1-byte: 1ms vs 100ms dominates 1Mbps vs 100Mbps
  - 25MB: 1Mbps vs 100Mbps dominates 1ms vs 100ms
- Infinite bandwidth
  - RTT dominates
    - $\text{Throughput} = \text{TransferSize} / \text{TransferTime}$
    - $\text{TransferTime} = \text{RTT} + 1/\text{Bandwidth} \times \text{TransferSize}$
  - 1-MB *file* to 1-Gbps link as 1-KB *packet* to 1-Mbps link

# Delay x Bandwidth Product

- Amount of data “in flight” or “in the pipe”
- Usually relative to RTT
- Example:  $100\text{ms} \times 45\text{Mbps} = 560\text{KB}$



# Socket API

- Creating a socket

`int socket(int domain, int type, int protocol)`

- `domain = PF_INET, PF_UNIX`
- `type = SOCK_STREAM, SOCK_DGRAM, SOCK_RAW`

- Passive Open (on server)

`int bind(int socket, struct sockaddr *addr, int addr_len)`

`int listen(int socket, int backlog)`

`int accept(int socket, struct sockaddr *addr, int addr_len)`



## Sockets (cont)

- Active Open (on client)

```
int connect(int socket, struct sockaddr *addr,  
            int addr_len)
```

- Sending/Receiving Messages

```
int send(int socket, char *msg, int mlen, int flags)  
int recv(int socket, char *buf, int blen, int flags)
```

# Protocol-to-Protocol Interface

- Configure multiple layers
  - static versus extensible
- Process Model
  - avoid context switches
- Buffer Model
  - avoid data copies

# *Implementation Issues*

*Go To Talk Outline*

# Elements of a Protocol Implementation

## Outline

Service Interface

Process Model

Common Subroutines

Example Protocol

# Socket API

- Creating a socket

`int socket(int domain, int type, int protocol)`

- `domain = PF_INET, PF_UNIX`
- `type = SOCK_STREAM, SOCK_DGRAM, SOCK_RAW`

- Passive Open (on server)

`int bind(int socket, struct sockaddr *addr, int addr_len)`

`int listen(int socket, int backlog)`

`int accept(int socket, struct sockaddr *addr, int addr_len)`

# Sockets (cont)

- Active Open (on client)

```
int connect(int socket, struct sockaddr *addr,  
            int addr_len)
```

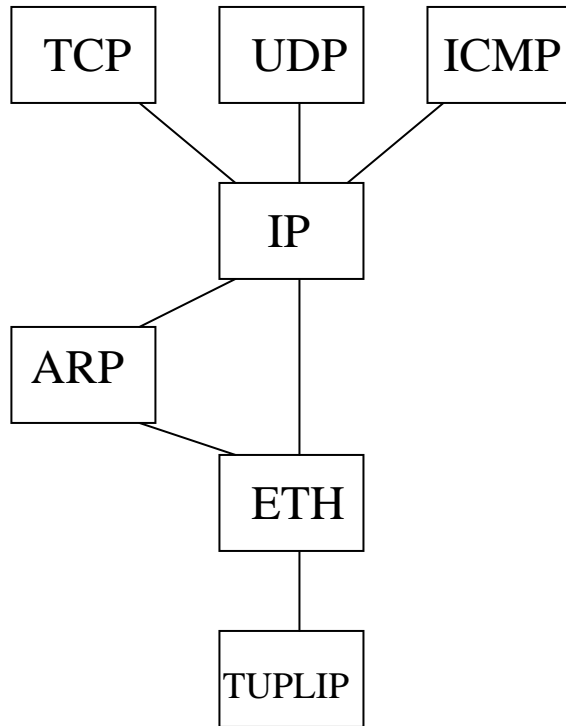
- Sending/Receiving Messages

```
int send(int socket, char *msg, int mlen, int flags)  
int recv(int socket, char *buf, int blen, int flags)
```

# Protocol-to-Protocol Interface

- Configure multiple layers
  - static versus extensible
- Process Model
  - avoid context switches
- Buffer Model
  - avoid data copies

# Configuration



```
name=tulip;  
name=eth protocols=tulip;  
name=arp protocols=eth;  
name=ip protocols=eth,arp;  
name=icmp protocols=ip;  
name=udp protocols=ip;  
name=tcp protocols=ip;
```



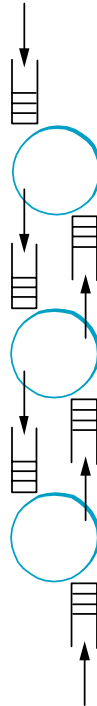
# Protocol Objects

- Active Open  
`Sessn xOpen(Prot1 hlp, Prot1 llp,  
Part *participants)`
- Passive Open  
`XkReturn xOpenEnable(Prot1 hlp, Prot1 llp,  
Part *participant)`  
  
`XkReturn xOpenDone(Prot1 hlp,  
Prot1 llp, Sessn session,  
Part *participants)`
- Demultiplexing  
`XkReturn xDemux(Prot1 hlp, Sessn lls,  
Msg *message)`

# Session Objects

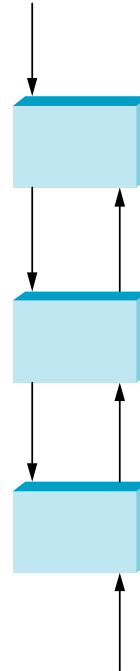
- Local end-point of a channel
- Interpret messages and maintain channel state
- Export operations for sending and receiving messages
- Operations
  - send a message  
**XkReturn xPush(Sessn lls,Msg \*message)**
  - deliver a message  
**XkReturn xPop(Sessn hls, Sessn lls,  
Msg \*message,void \*hdr)**

# Process Model



(a)

Process-per-Protocol

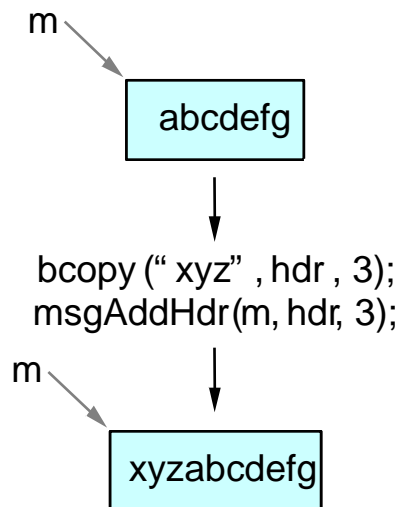


(b)

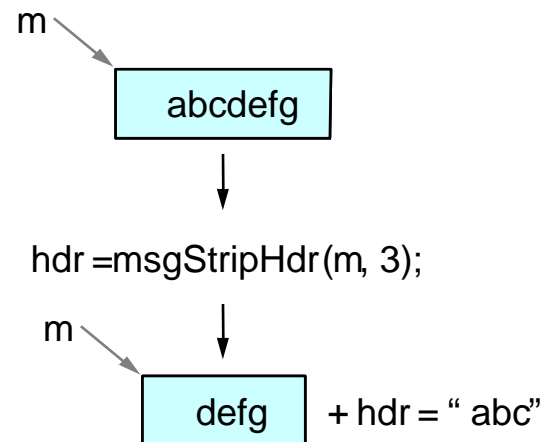
Process-per-Message

# Message Library

- Add header

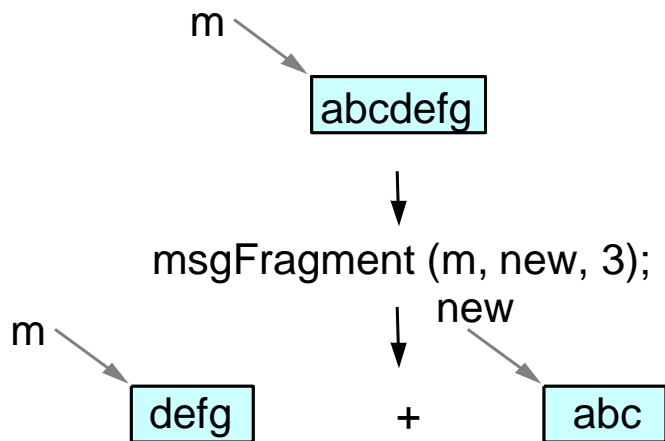


- Strip header

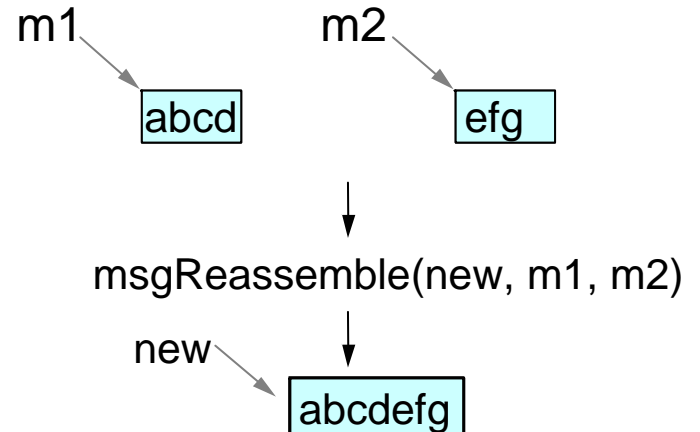


# Message Library (cont)

- Fragment message



- Reassemble messages



# Event Library

- Scheduling Timeouts & Book-keeping activity
- Operations

```
Event evSchedule(EvFunc function,  
                void *argument,  
                int time)
```

```
EvCancelReturn evCancel(Event event)
```

# Map Library

- Demultiplex Packets
- Operations

```
Map mapCreate(int number,int size)
```

```
Binding mapBind(Map map,  
                void *key,  
                void *id)
```

```
XkReturn mapResolve(Map map,  
                    void *key,void **id)
```

# Example

```
static Sessn
aspOpen(Prot1 self, Prot1 hlp, Part *p)
{
    Sessn      asp_s;
    Sessn      lls;
    ActiveId   key;
    Prot1State *pstate = (Prot1State *)self->state;

    bzero((char *)&key, sizeof(key));

    /* High level protocol must specify both */
    /* local and remote ASP port */
    key.localport = *((AspPort *) partPop(p[0]));
    key.remoteport = *((AspPort *) partPop(p[1]));
    /* Assume failure until proven otherwise */
    asp_s = XK_FAILURE;
```



```

/* Open session on low-level protocol */
lls = xOpen(self, xGetDown(self, 0), p);
if ( lls != XK_FAILURE )
{
    key.lls = lls;
    /* Check for session in active map */
    if (mapResolve(pstate->activemap, &key,
        (void **)&asp_s) == XK_FAILURE)
    {
        /* Session not in map; initialize it */
        asp_s = asp_init_sessn(self, hlp, &key);
        if ( asp_s != XK_FAILURE )
        {
            /* A successful open */
            return asp_s;
        }
    }

    /* Error has occurred */
    xClose(lls);
}
return asp_s;
}

```

```

static XkReturn aspPush(Sessn s, Msg *msg)
{
    SessnState *sstate;
    AspHdr      hdr;
    void        *buf;

    /* create a header */
    sstate = (SessnState *) s->state;
    hdr = sstate->hdr;
    hdr.ulen = msgLen(msg) + HLEN;

    /* attach header and send */
    buf = msgAddrHdr(msg, HLEN);
    aspHdrStore(&hdr, buf, HLEN, msg);
    return xPush(xGetDown(s, 0), msg);
}

```

```

static XkReturn aspDemux(Prot1 self, Sessn lls, Msg *msg)
{
    AspHdr      h;
    Sessn       s;
    ActiveId    activeid;
    PassiveId   passiveid;
    Prot1State  *pstate;
    Enable      *e;
    void        *buf;

    pstate = (Prot1State *)self->state;

    /* extract the header from the message */
    buf = msgStripHdr(msg, HLEN);
    aspHdrLoad(&h, buf, HLEN, msg);

    /* construct a demux key from the header */
    bzero((char *)&activeid, sizeof(activeid));
    activeid.localport = h.dport;
    activeid.remoteport = h.sport;
    activeid.lls = lls;

```

```

/* see if demux key is in the active map */
if (mapResolve(pstate->activemap, &activeid,
    (void **)&s) == XK_FAILURE)
{
    /* didn't find an active session */
    /* so check passive map */
    passiveid = h.dport;
    if (mapResolve(pstate->passivemap, &passiveid,
        (void **)&e) == XK_FAILURE)
    {
        /* drop the message */
        return XK_SUCCESS;
    }

    /* port was enabled, so create a new */
    /* session and inform hlp */
    s = asp_init_sessn(self, e->hlp, &activeid);
    xOpenDone(e->hlp, s, self);
}
/* pop the message to the session */
return xPop(s, lls, msg, &h);

```

```

}

```

```

static XkReturn
aspPop(Sessn s, Sessn ds, Msg *msg, void *inHdr)
{
    AspHdr      *h = (AspHdr *) inHdr;

    /* truncate message to length shown in header */
    if ((h->ulen - HLEN) < msgLen(msg))
        msgTruncate(msg, (int) h->ulen);

    /* pass message up the protocol stack */
    return xDemux(xGetUp(s), s, msg);
}

```

## *Point-to-Point Links*

*Go To Talk Outline*

# Point-to-Point Links

## Outline

- Encoding

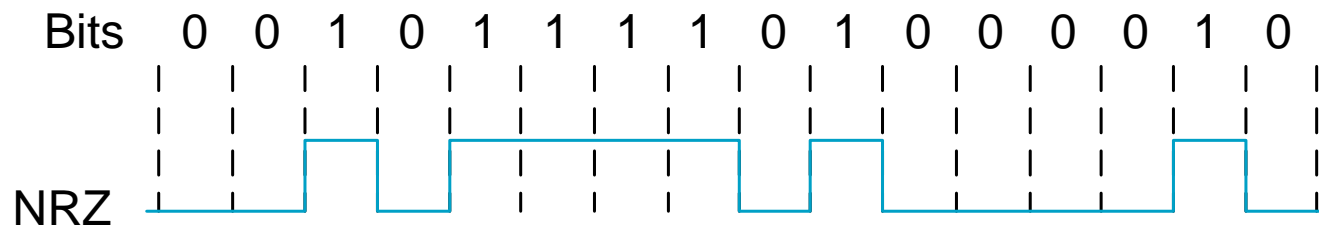
- Framing

- Error Detection

- Sliding Window Algorithm

# Encoding

- Signals propagate over a physical medium
  - modulate electromagnetic waves
  - e.g., vary voltage
- Encode binary data onto signals
  - e.g., 0 as low signal and 1 as high signal
  - known as Non-Return to zero (NRZ)





# Problem: Consecutive 1s or 0s

- Low signal (0) may be interpreted as no signal
- High signal (1) leads to baseline wander
- Unable to recover clock

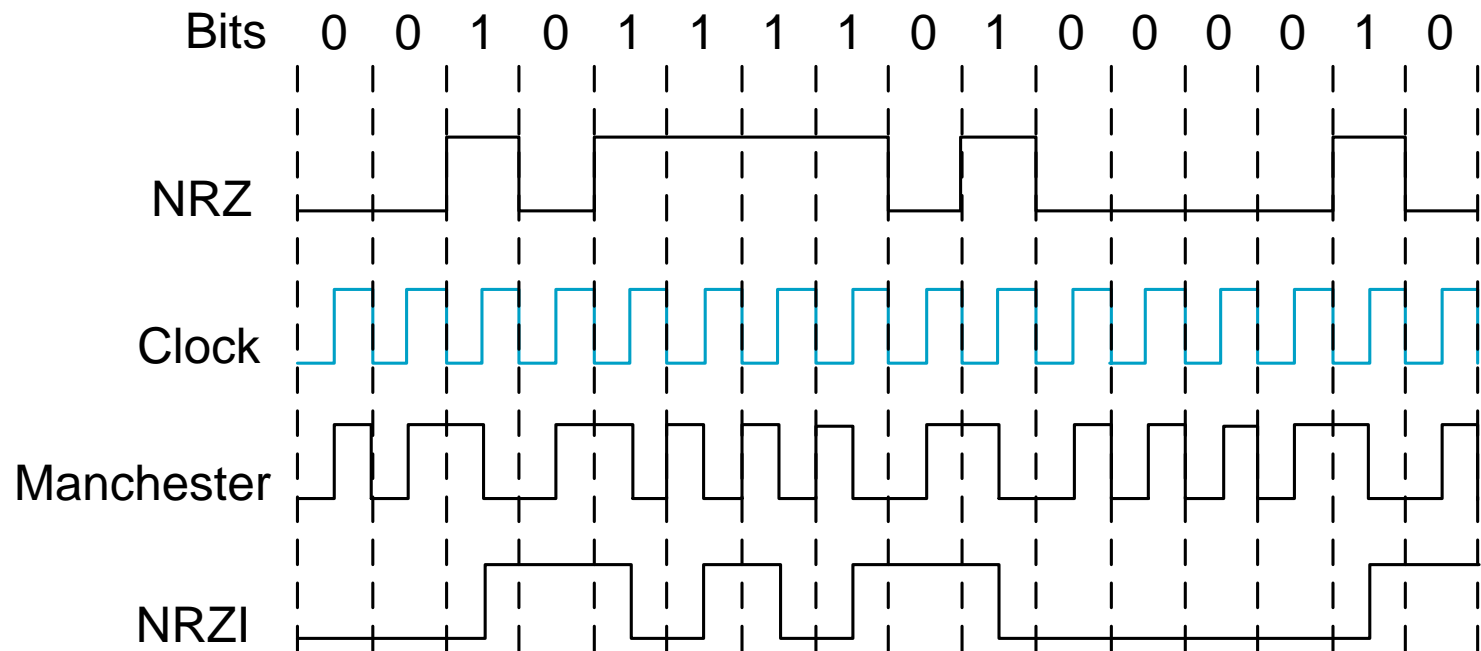
# Alternative Encodings

- Non-return to Zero Inverted (NRZI)
  - make a transition from current signal to encode a one; stay at current signal to encode a zero
  - solves the problem of consecutive ones
- Manchester
  - transmit XOR of the NRZ encoded data and the clock
  - only 50% efficient (bit rate =  $1/2$  baud rate)

# Encodings (cont)

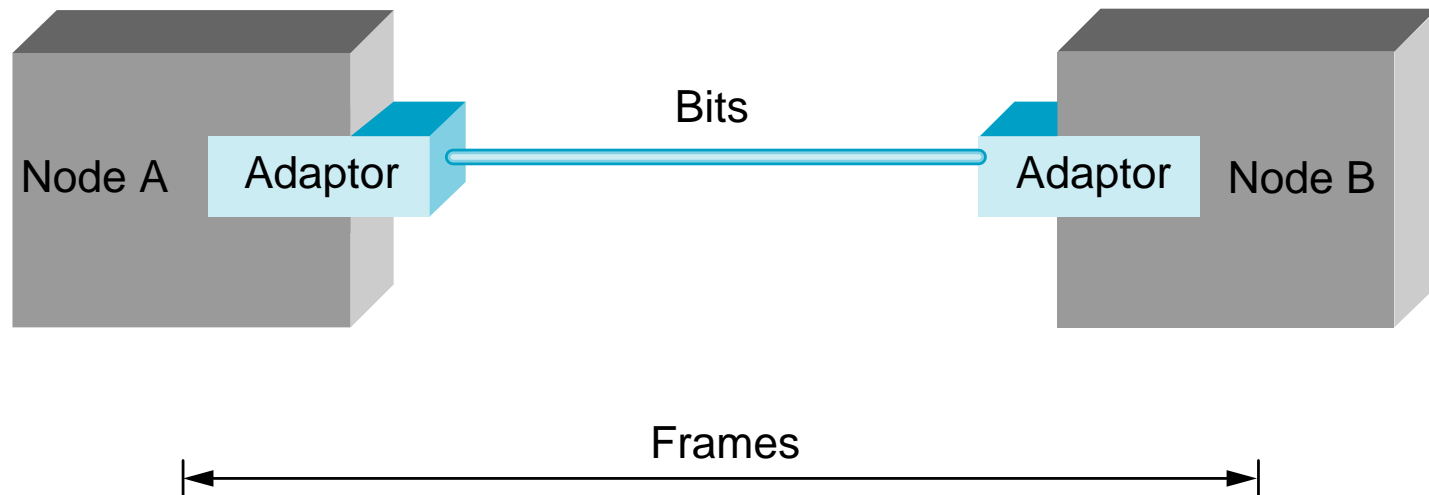
- 4B/5B
  - every 4 bits of data encoded in a 5-bit code
  - 5-bit codes selected to have no more than one leading 0 and no more than two trailing 0s
  - thus, never get more than three consecutive 0s
  - resulting 5-bit codes are transmitted using NRZI
  - achieves 80% efficiency

# Encodings (cont)



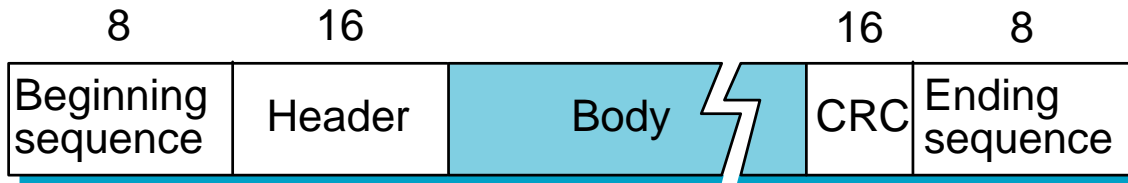
# Framing

- Break sequence of bits into a frame
- Typically implemented by network adaptor



# Approaches

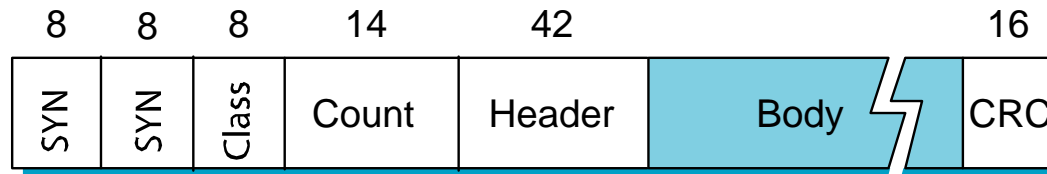
- Sentinel-based
  - delineate frame with special pattern: 01111110
  - e.g., HDLC, SDLC, PPP



- problem: special pattern appears in the payload
- solution: *bit stuffing*
  - sender: insert 0 after five consecutive 1s
  - receiver: delete 0 that follows five consecutive 1s

# Approaches (cont)

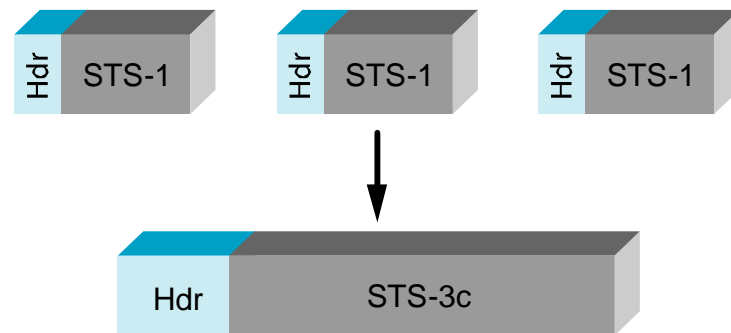
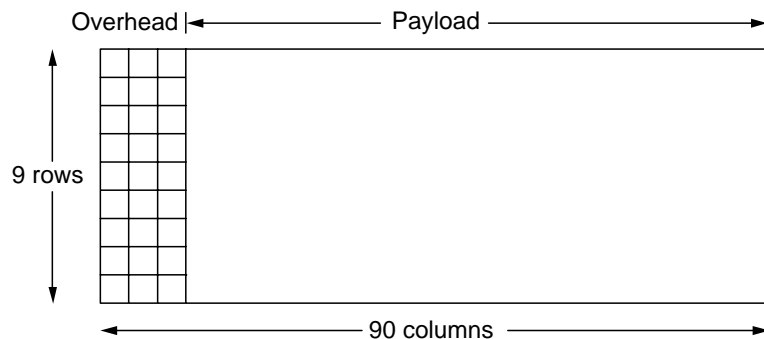
- Counter-based
  - include payload length in header
  - e.g., DDCMP



- problem: count field corrupted
- solution: catch when CRC fails

# Approaches (cont)

- Clock-based
  - each frame is 125us long
  - e.g., SONET: Synchronous Optical Network
  - STS- $n$  (STS-1 = 51.84 Mbps)





# Cyclic Redundancy Check

- Add  $k$  bits of redundant data to an  $n$ -bit message
  - want  $k \ll n$
  - e.g.,  $k = 32$  and  $n = 12,000$  (1500 bytes)
- Represent  $n$ -bit message as  $n-1$  degree polynomial
  - e.g., MSG=10011010 as  $M(x) = x^7 + x^4 + x^3 + x^1$
- Let  $k$  be the degree of some divisor polynomial
  - e.g.,  $C(x) = x^3 + x^2 + 1$

## CRC (cont)

- Transmit polynomial  $P(x)$  that is evenly divisible by  $C(x)$ 
  - shift left  $k$  bits, i.e.,  $M(x)x^k$
  - subtract remainder of  $M(x)x^k / C(x)$  from  $M(x)x^k$
- Receiver polynomial  $P(x) + E(x)$ 
  - $E(x) = 0$  implies no errors
- Divide  $(P(x) + E(x))$  by  $C(x)$ ; remainder zero if:
  - $E(x)$  was zero (no error), or
  - $E(x)$  is exactly divisible by  $C(x)$

# Selecting $C(x)$

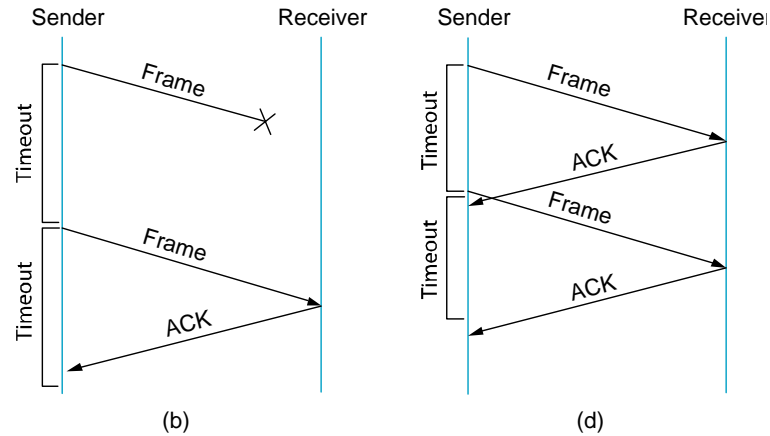
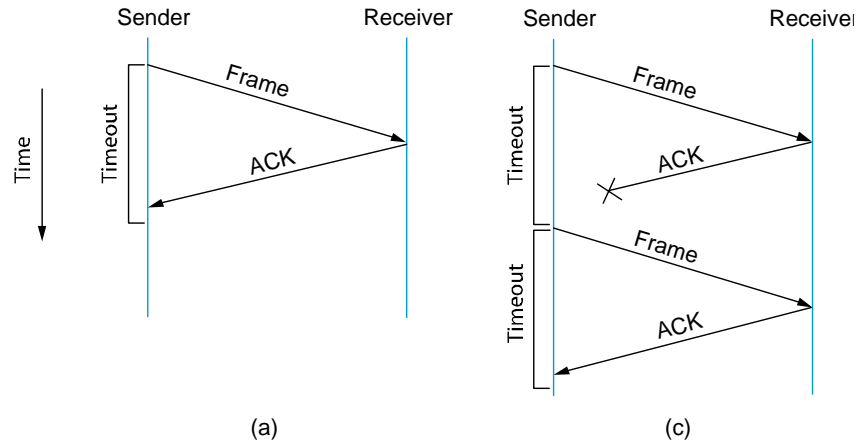
- All single-bit errors, as long as the  $x^k$  and  $x^0$  terms have non-zero coefficients.
- All double-bit errors, as long as  $C(x)$  contains a factor with at least three terms
- Any odd number of errors, as long as  $C(x)$  contains the factor  $(x + 1)$
- Any ‘burst’ error (i.e., sequence of consecutive error bits) for which the length of the burst is less than  $k$  bits.
- Most burst errors of larger than  $k$  bits can also be detected
- See Table 2.6 on page 102 for common  $C(x)$

# Internet Checksum Algorithm

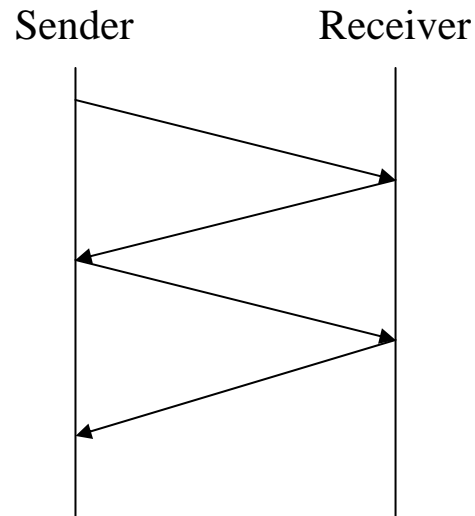
- View message as a sequence of 16-bit integers; sum using 16-bit ones-complement arithmetic; take ones-complement of the result.

```
u_short
cksum(u_short *buf, int count)
{
    register u_long sum = 0;
    while (count--)
    {
        sum += *buf++;
        if (sum & 0xFFFF0000)
        {
            /* carry occurred, so wrap around */
            sum &= 0xFFFF;
            sum++;
        }
    }
    return ~(sum & 0xFFFF);
}
```

# Acknowledgements & Timeouts



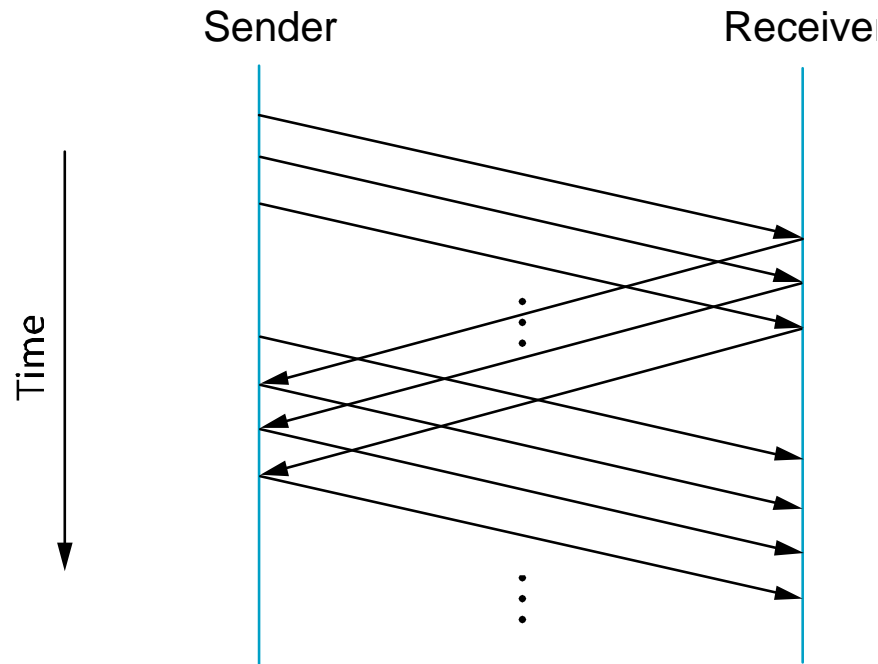
# Stop-and-Wait



- Problem: keeping the pipe full
- Example
  - $1.5\text{Mbps link} \times 45\text{ms RTT} = 67.5\text{Kb (8KB)}$
  - 1KB frames implies 1/8th link utilization

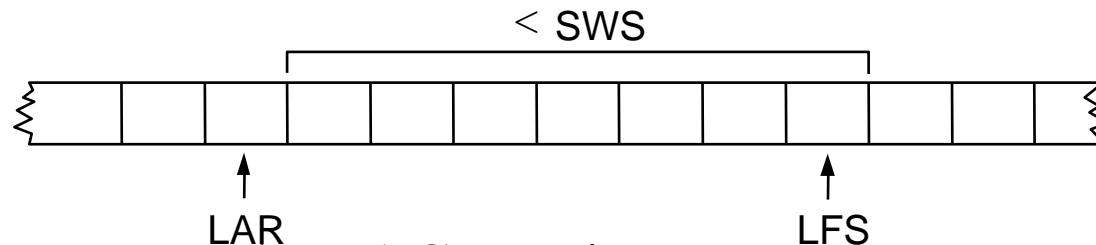
# Sliding Window

- Allow multiple outstanding (un-ACKed) frames
- Upper bound on un-ACKed frames, called *window*



# SW: Sender

- Assign sequence number to each frame (**SeqNum**)
- Maintain three state variables:
  - send window size (**SWS**)
  - last acknowledgment received (**LAR**)
  - last frame sent (**LFS**)
- Maintain invariant: **LFS** - **LAR**  $\leq$  **SWS**



- Advance **LAR** when ACK arrives
- Buffer up to **sws** frames



# SW: Receiver

- Maintain three state variables
  - receive window size (**RWS**)
  - largest frame acceptable (**LFA**)
  - last frame received (**LFR**)
- Maintain invariant: **LFA - LFR**  $\leq$  **RWS**



- Frame **SeqNum** arrives:
  - if **LFR**  $<$  **SeqNum**  $\leq$  **LFA**  $\longrightarrow$  accept
  - if **SeqNum**  $\leq$  **LFR** or **SeqNum**  $>$  **LFA**  $\longrightarrow$  discarded
- Send cumulative ACKs

# Sequence Number Space

- **SeqNum** field is finite; sequence numbers wrap around
- Sequence number space must be larger than number of outstanding frames
- **SWS  $\leq$  MaxSeqNum-1** is not sufficient
  - suppose 3-bit **SeqNum** field (0..7)
  - **SWS=RWS=7**
  - sender transmit frames 0..6
  - arrive successfully, but ACKs lost
  - sender retransmits 0..6
  - receiver expecting 7, 0..5, but receives second incarnation of 0..5
- **SWS  $< (\text{MaxSeqNum}+1) / 2$**  is correct rule
- Intuitively, **SeqNum** “slides” between two halves of sequence number space

# Concurrent Logical Channels

- Multiplex 8 logical channels over a single link
- Run stop-and-wait on each logical channel
- Maintain three state bits per channel
  - channel busy
  - current sequence number out
  - next sequence number in
- Header: 3-bit channel num, 1-bit sequence num
  - 4-bits total
  - same as sliding window protocol
- Separates *reliability* from *order*

# *Shared Media Networks*

*Go To Talk Outline*

# Shared Access Networks

## Outline

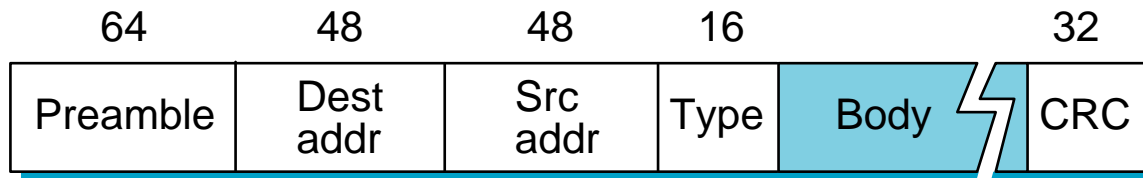
- Bus (Ethernet)

- Token ring (FDDI)

- Wireless (802.11)

# Ethernet Overview

- History
  - developed by Xerox PARC in mid-1970s
  - roots in Aloha packet-radio network
  - standardized by Xerox, DEC, and Intel in 1978
  - similar to IEEE 802.3 standard
- CSMA/CD
  - carrier sense
  - multiple access
  - collision detection
- Frame Format



# Ethernet (cont)

- Addresses
  - unique, 48-bit unicast address assigned to each adapter
  - example: **8:0:e4:b1:2**
  - broadcast: all **1**s
  - multicast: first bit is **1**
- Bandwidth: 10Mbps, 100Mbps, 1Gbps
- Length: 2500m (500m segments with 4 repeaters)
- Problem: Distributed algorithm that provides fair access

# Transmit Algorithm

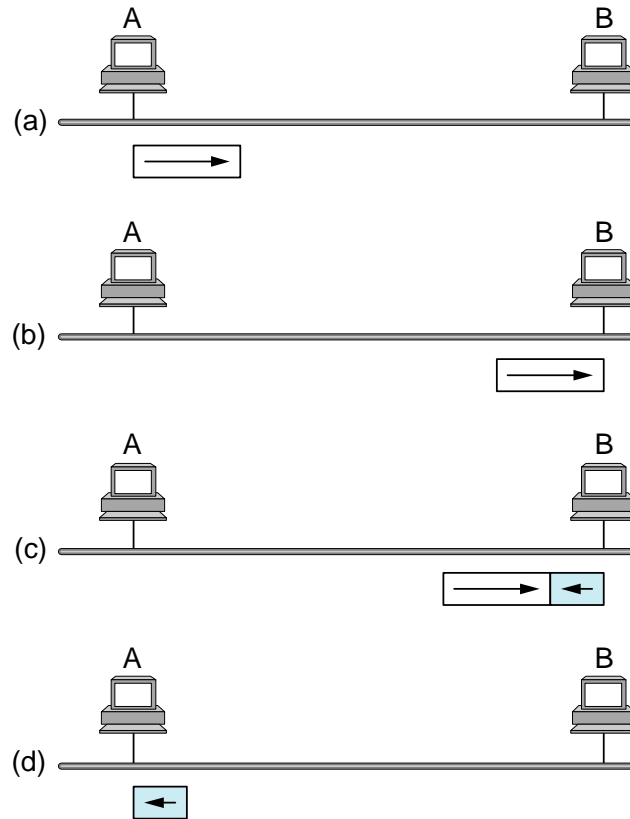
- If line is idle...
  - send immediately
  - upper bound message size of 1500 bytes
  - must wait 9.6us between back-to-back frames
- If line is busy...
  - wait until idle and transmit immediately
  - called *1-persistent* (special case of *p-persistent*)



# Algorithm (cont)

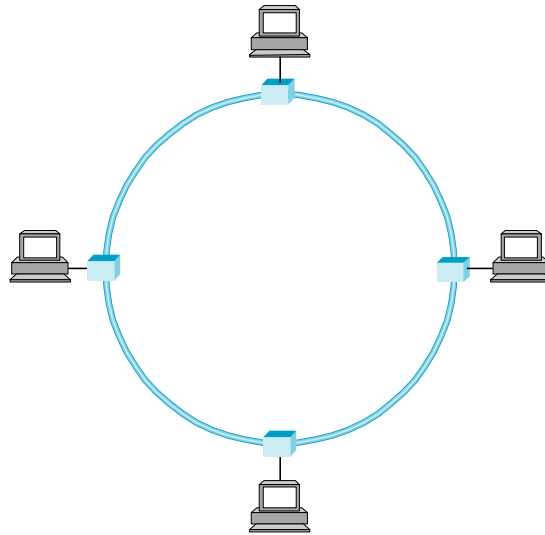
- If collision...
  - jam for 32 bits, then stop transmitting frame
  - minimum frame is 64 bytes (header + 46 bytes of data)
  - delay and try again
    - 1st time: 0 or 51.2us
    - 2nd time: 0, 51.2, or 102.4us
    - 3rd time: 51.2, 102.4, or 153.6us
    - $n$ th time:  $k \times 51.2\text{us}$ , for randomly selected  $k=0..2^n - 1$
    - give up after several tries (usually 16)
    - exponential backoff

# Collisions



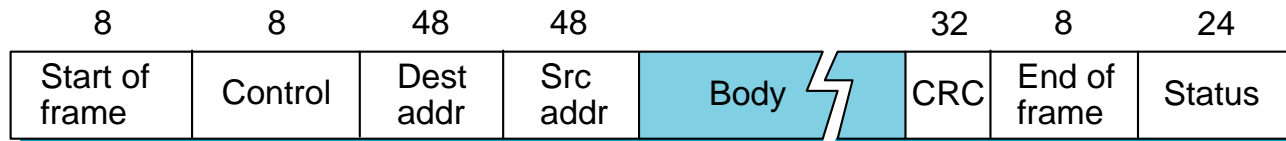
# Token Ring Overview

- Examples
  - 16Mbps IEEE 802.5 (based on earlier IBM ring)
  - 100Mbps Fiber Distributed Data Interface (FDDI)



# Token Ring (cont)

- Idea
  - Frames flow in one direction: upstream to downstream
  - special bit pattern (token) rotates around ring
  - must capture token before transmitting
  - release token after done transmitting
    - immediate release
    - delayed release
  - remove your frame when it comes back around
  - stations get round-robin service
- Frame Format



# Timed Token Algorithm

- Token Holding Time (THT)
  - upper limit on how long a station can hold the token
- Token Rotation Time (TRT)
  - how long it takes the token to traverse the ring
  - **$TRT \leq \text{ActiveNodes} \times THT + \text{RingLatency}$**
- Target Token Rotation Time (TTRT)
  - agreed-upon upper bound on TRT

## Algorithm (cont)

- Each node measures TRT between successive tokens
  - if measured-TRT  $>$  TTRT: token is late so don't send
  - if measured-TRT  $<$  TTRT: token is early so OK to send
- Two classes of traffic
  - synchronous: can always send
  - asynchronous: can send only if token is early
- Worse case:  $2 \times \text{TTRT}$  between seeing token
- Back-to-back  $2 \times \text{TTRT}$  rotations not possible

# Token Maintenance

- Lost Token
  - no token when initializing ring
  - bit error corrupts token pattern
  - node holding token crashes
- Generating a Token (and agreeing on TTRT)
  - execute when join ring or suspect a failure
  - send a *claim frame* that includes the node's TTRT *bid*
  - when receive claim frame, update the bid and forward
  - if your claim frame makes it all the way around the ring:
    - your bid was the lowest
    - everyone knows TTRT
    - you insert new token

# Maintenance (cont)

- Monitoring for a Valid Token
  - should periodically see valid transmission (frame or token)
  - maximum gap = ring latency + max frame  $\leq 2.5\text{ms}$
  - set timer at 2.5ms and send claim frame if it fires



# Wireless LANs

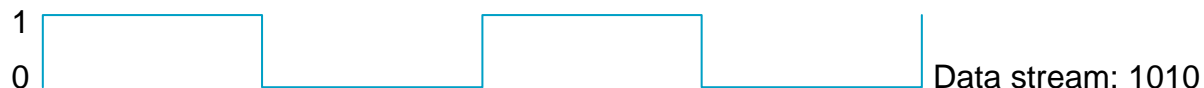
- IEEE 802.11
- Bandwidth: 1 - 11 Mbps
- Physical Media
  - spread spectrum radio (2.4GHz)
  - diffused infrared (10m)

# Spread Spectrum

- Idea
  - spread signal over wider frequency band than required
  - originally designed to thwart jamming
- Frequency Hopping
  - transmit over random sequence of frequencies
  - sender and receiver share...
    - pseudorandom number generator
    - seed
  - 802.11 uses 79 x 1MHz-wide frequency bands

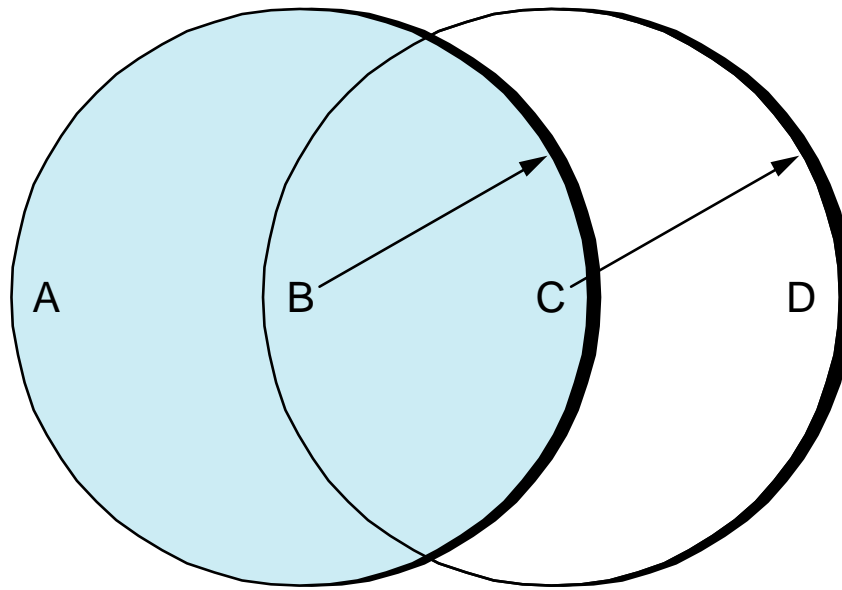
# Spread Spectrum (cont)

- Direct Sequence
  - for each bit, send XOR of that bit and  $n$  random bits
  - random sequence known to both sender and receiver
  - called  $n$ -bit *chipping code*
  - 802.11 defines an 11-bit chipping code



# Collisions Avoidance

- Similar to Ethernet
- Problem: *hidden* and *exposed* nodes

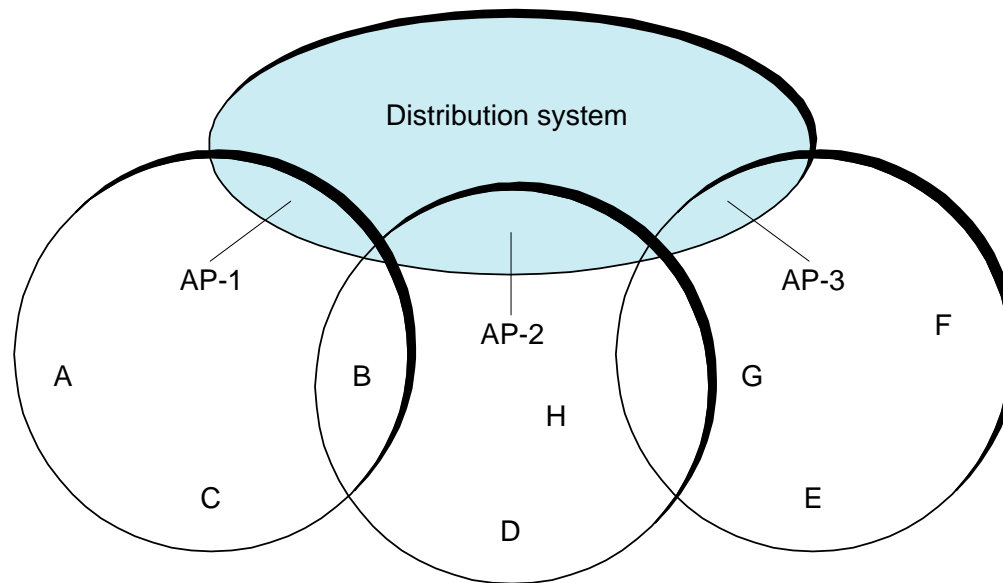


# MACAW

- Sender transmits **RequestToSend** (RTS) frame
- Receiver replies with **ClearToSend** (CTS) frame
- Neighbors...
  - see CTS: keep quiet
  - see RTS but not CTS: ok to transmit
- Receiver sends **ACK** when has frame
  - neighbors silent until see ACK
- Collisions
  - no collisions detection
  - known when don't receive CTS
  - exponential backoff

# Supporting Mobility

- Case 1: *ad hoc* networking
- Case 2: *access points* (AP)
  - tethered
  - each mobile node associates with an AP



# Mobility (cont)

- Scanning (selecting an AP)
  - node sends **Probe** frame
  - all AP's w/in reach reply with **ProbeResponse** frame
  - node selects one AP; sends it **AssociateRequest** frame
  - AP replies with **AssociationResponse** frame
  - new AP informs old AP via tethered network
- When
  - active: when join or move
  - passive: AP periodically sends **Beacon** frame

# *Switched Networks (Packet Switching)*

*Go To Talk Outline*



# Switching and Forwarding

## Outline

- Store-and-Forward Switches

- Bridges and Extended LANs

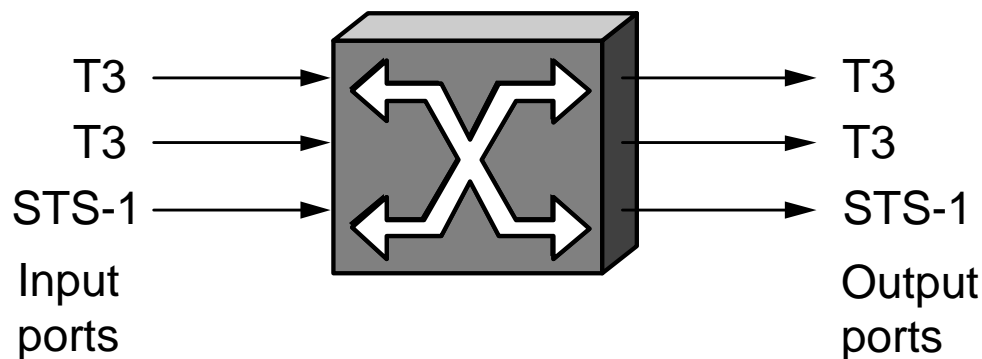
- Cell Switching

- Segmentation and Reassembly

# Scalable Networks

- Switch

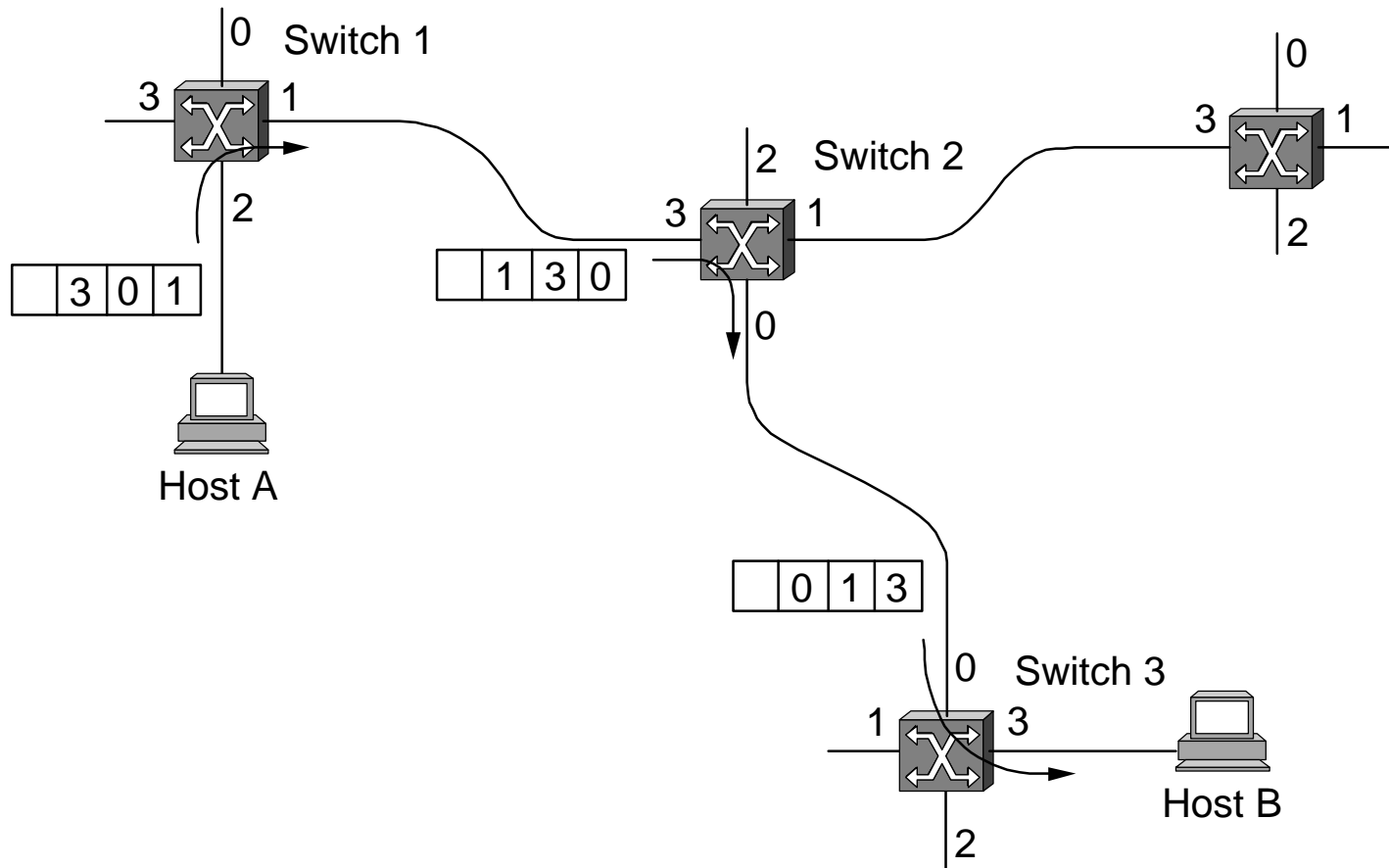
- forwards packets from input port to output port
- port selected based on address in packet header



- Advantages

- cover large geographic area (tolerate latency)
- support large numbers of hosts (scalable bandwidth)

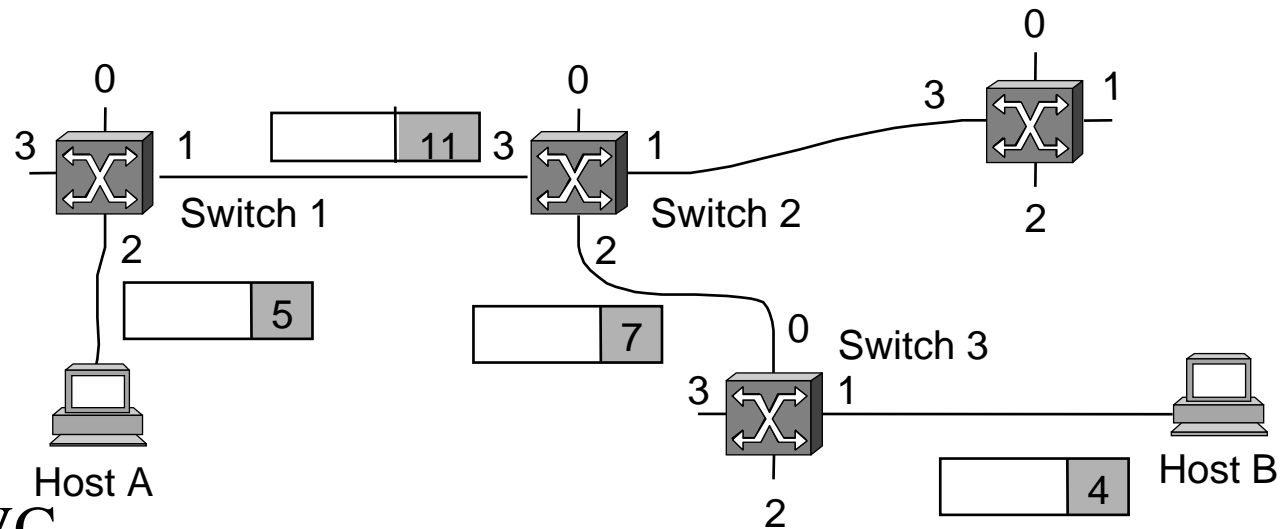
# Source Routing



# Virtual Circuit Switching

- Explicit connection setup (and tear-down) phase
- Subsequence packets follow same circuit
- Sometimes called *connection-oriented* model

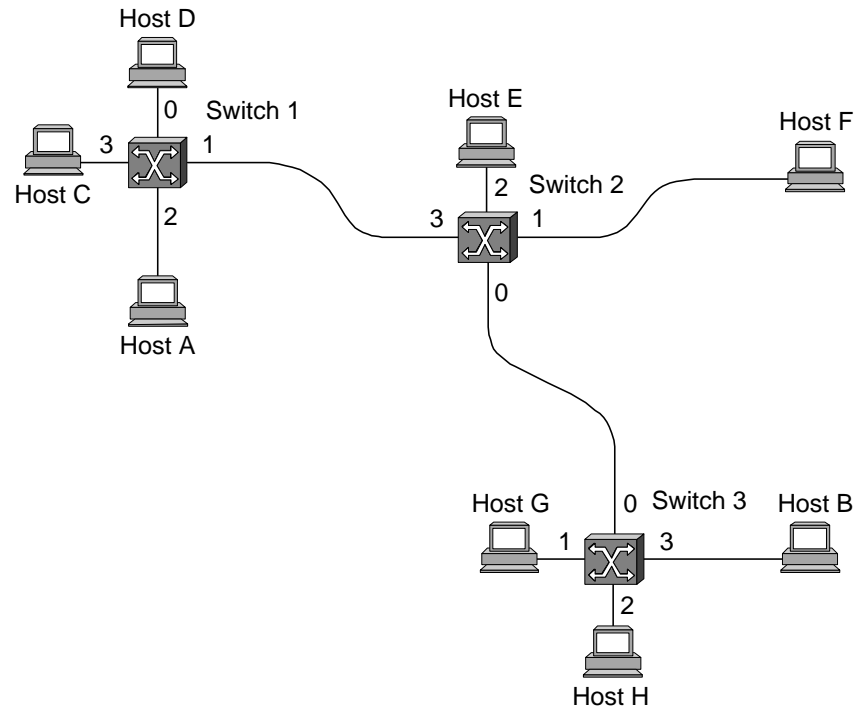
- Analogy:  
phone call
- Each switch  
maintains a VC  
table



# Datagram Switching

- No connection setup phase
- Each packet forwarded independently
- Sometimes called *connectionless* model

- Analogy: postal system
- Each switch maintains a forwarding (routing) table



# Example Tables

- Circuit Table  
(switch 1, port 2)

VC In	VC Out	Port Out
5	11	1
6	8	1
...	...	...

- Forwarding Table  
(switch 1)

Address	Port
A	2
C	3
F	1
G	1
...	...

# Virtual Circuit Model

- Typically wait full RTT for connection setup before sending first data packet.
- While the connection request contains the full address for destination, each data packet contains only a small identifier, making the per-packet header overhead small.
- If a switch or a link in a connection fails, the connection is broken and a new one needs to be established.
- Connection setup provides an opportunity to reserve resources.

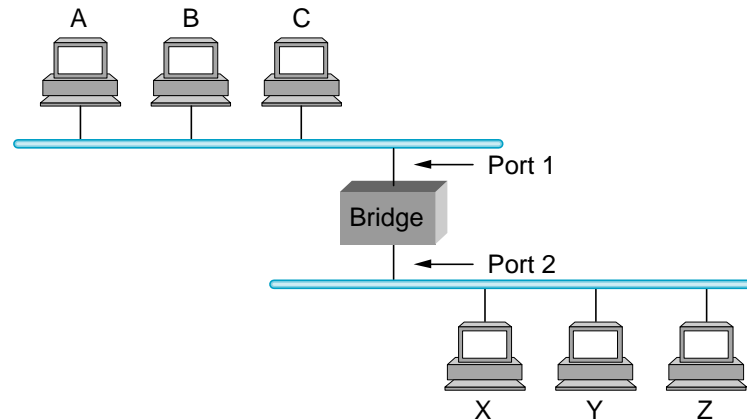
# Datagram Model

- There is no round trip delay waiting for connection setup; a host can send data as soon as it is ready.
- Source host has no way of knowing if the network is capable of delivering a packet or if the destination host is even up.
- Since packets are treated independently, it is possible to route around link and node failures.
- Since every packet must carry the full address of the destination, the overhead per packet is higher than for the connection-oriented model.



# Bridges and Extended LANs

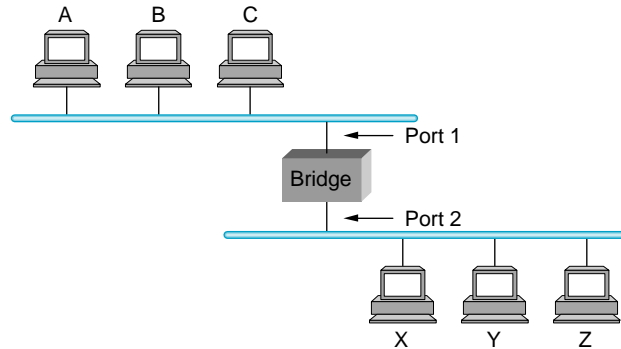
- LANs have physical limitations (e.g., 2500m)
- Connect two or more LANs with a *bridge*
  - accept and forward strategy
  - level 2 connection (does not add packet header)



- Ethernet Switch = Bridge on Steroids

# Learning Bridges

- Do not forward when unnecessary
- Maintain forwarding table

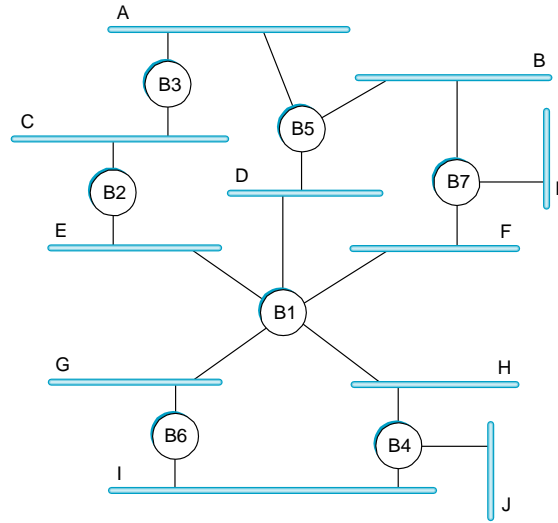


Host	Port
A	1
B	1
C	1
X	2
Y	2
Z	2

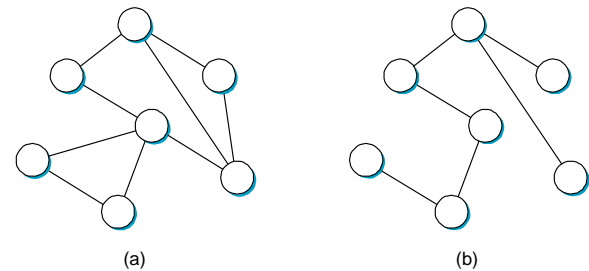
- Learn table entries based on source address
- Table is an optimization; need not be complete
- Always forward broadcast frames

# Spanning Tree Algorithm

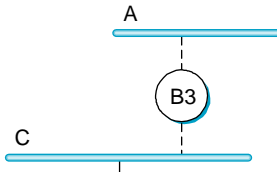
- Problem: loops

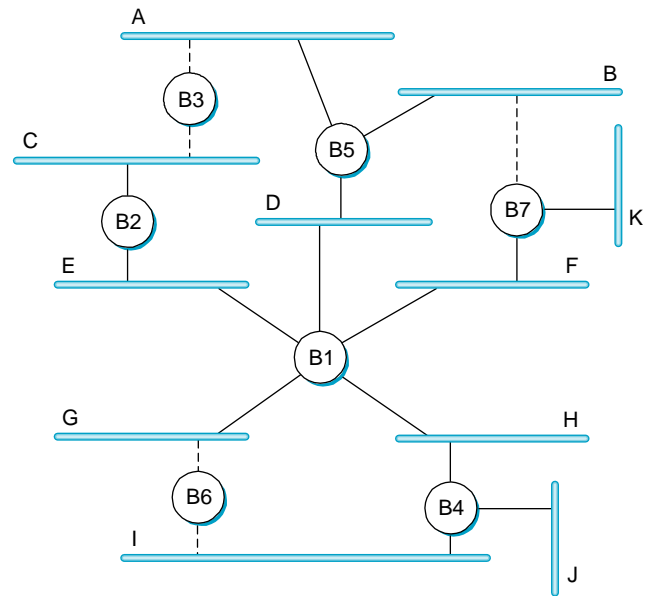


- Bridges run a distributed spanning tree algorithm
  - select which bridges actively forward
  - developed by Radia Perlman
  - now IEEE 802.1 specification



# Algorithm Overview

- Each bridge has unique id (e.g., B1, B2, B3)
  - Select bridge with smallest id as root
  - Select bridge on each LAN closest to root as designated bridge (use id to break ties)
  - Each bridge forwards frames over each LAN for which it is the designated bridge
- 
- The diagram shows a central bridge labeled B3, represented by a circle with a blue border. It is connected to two horizontal blue lines representing LANs. The top LAN is labeled 'A' and the bottom LAN is labeled 'C'. Dashed vertical lines connect the bridge B3 to both LANs A and C.



# Algorithm Details

- Bridges exchange configuration messages
  - id for bridge sending the message
  - id for what the sending bridge believes to be root bridge
  - distance (hops) from sending bridge to root bridge
- Each bridge records current best configuration message for each port
- Initially, each bridge believes it is the root

# Algorithm Detail (cont)

- When learn not root, stop generating config messages
  - in steady state, only root generates configuration messages
- When learn not designated bridge, stop forwarding config messages
  - in steady state, only designated bridges forward config messages
- Root continues to periodically send config messages
- If any bridge does not receive config message after a period of time, it starts generating config messages claiming to be the root

# Broadcast and Multicast

- Forward all broadcast/multicast frames
  - current practice
- Learn when no group members downstream
- Accomplished by having each member of group G send a frame to bridge multicast address with G in source field

# Limitations of Bridges

- Do not scale
  - spanning tree algorithm does not scale
  - broadcast does not scale
- Do not accommodate heterogeneity
- Caution: beware of transparency



# Cell Switching (ATM)

- Connection-oriented packet-switched network
- Used in both WAN and LAN settings
- Signaling (connection setup) Protocol: Q.2931
- Specified by ATM forum
- Packets are called *cells*
  - 5-byte header + 48-byte payload
- Commonly transmitted over SONET
  - other physical layers possible

# Variable vs Fixed-Length Packets

- No Optimal Length
  - if small: high header-to-data overhead
  - if large: low utilization for small messages
- Fixed-Length Easier to Switch in Hardware
  - simpler
  - enables parallelism

# Big vs Small Packets

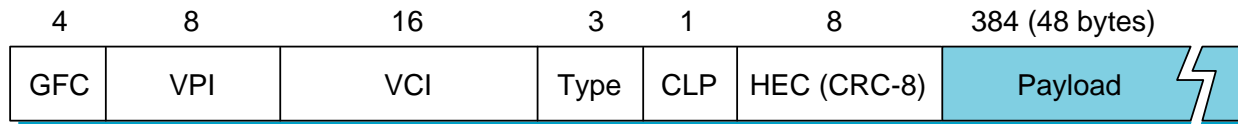
- Small Improves Queue behavior
  - finer-grained preemption point for scheduling link
    - maximum packet = 4KB
    - link speed = 100Mbps
    - transmission time =  $4096 \times 8/100 = 327.68\mu\text{s}$
    - high priority packet may sit in the queue 327.68us
    - in contrast,  $53 \times 8/100 = 4.24\mu\text{s}$  for ATM
  - near cut-through behavior
    - two 4KB packets arrive at same time
    - link idle for 327.68us while both arrive
    - at end of 327.68us, still have 8KB to transmit
    - in contrast, can transmit first cell after 4.24us
    - at end of 327.68us, just over 4KB left in queue

## Big vs Small (cont)

- Small Improves Latency (for voice)
  - voice digitally encoded at 64KBps (8-bit samples at 8KHz)
  - need full cell's worth of samples before sending cell
  - example: 1000-byte cells implies 125ms per cell (too long)
  - smaller latency implies no need for echo cancellers
- ATM Compromise: 48 bytes =  $(32+64)/2$

# Cell Format

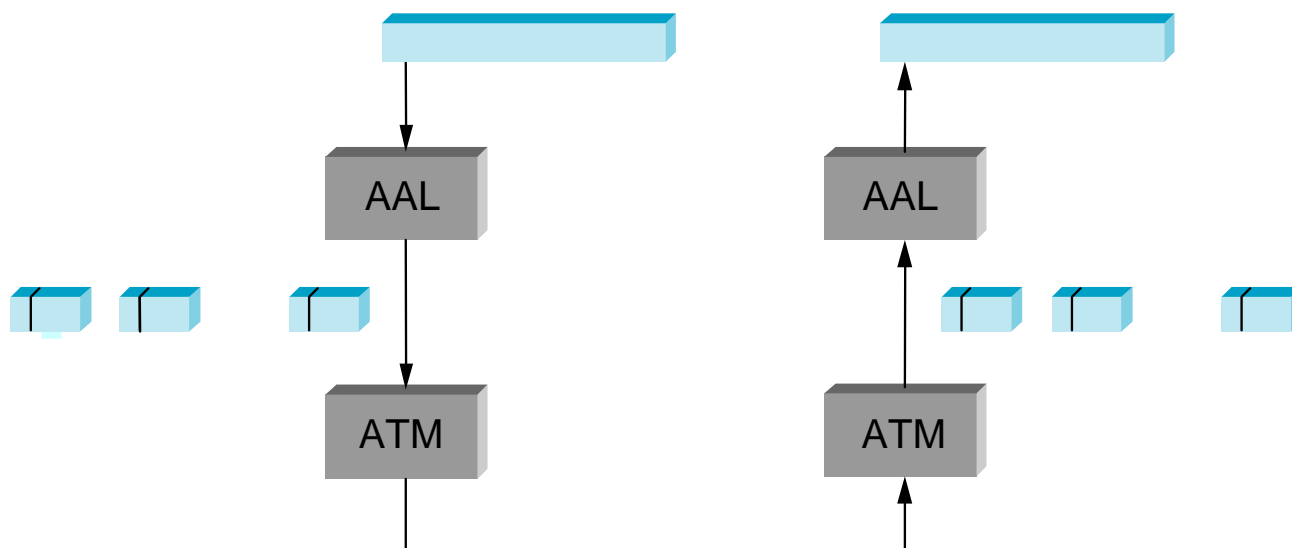
- User-Network Interface (UNI)



- host-to-switch format
  - GFC: Generic Flow Control (still being defined)
  - VCI: Virtual Circuit Identifier
  - VPI: Virtual Path Identifier
  - Type: management, congestion control, AAL5 (later)
  - CLPL Cell Loss Priority
  - HEC: Header Error Check (CRC-8)
- Network-Network Interface (NNI)
    - switch-to-switch format
    - GFC becomes part of VPI field

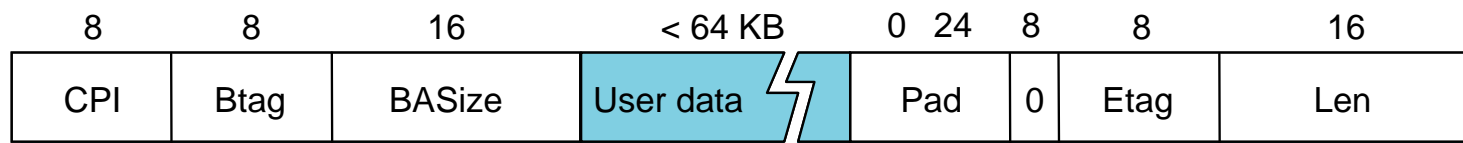
# Segmentation and Reassembly

- ATM Adaptation Layer (AAL)
  - AAL 1 and 2 designed for applications that need guaranteed rate (e.g., voice, video)
  - AAL 3/4 designed for packet data
  - AAL 5 is an alternative standard for packet data



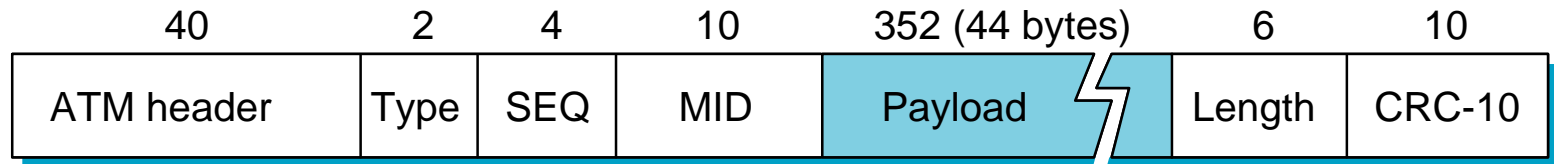
# AAL 3/4

- Convergence Sublayer Protocol Data Unit (CS-PDU)



- CPI: commerce part indicator (version field)
- Btag/Etag: beginning and ending tag
- BAsize: hint on amount of buffer space to allocate
- Length: size of whole PDU

# Cell Format

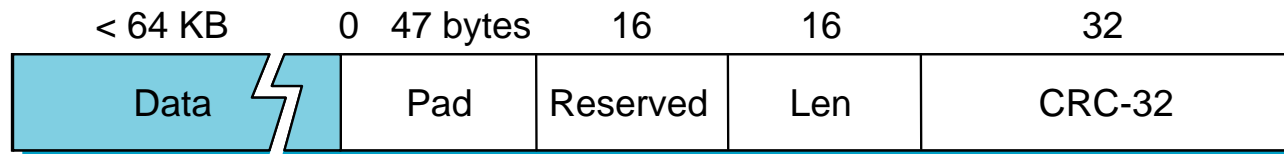


- Type
  - BOM: beginning of message
  - COM: continuation of message
  - EOM end of message
- SEQ: sequence of number
- MID: message id
- Length: number of bytes of PDU in this cell



# AAL5

- CS-PDU Format



- pad so trailer always falls at end of ATM cell
  - Length: size of PDU (data only)
  - CRC-32 (detects missing or misordered cells)
- Cell Format
    - end-of-PDU bit in Type field of ATM header

# *IP and the Internet (Internetworking)*

*Go To Talk Outline*

# Internetworking

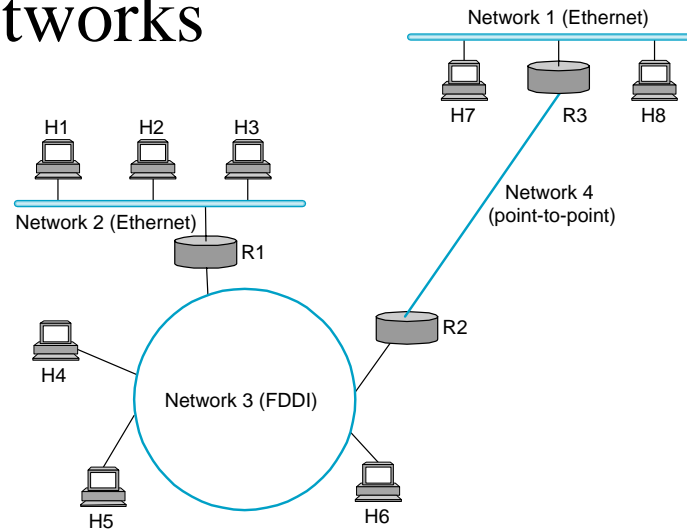
## Outline

Best Effort Service Model

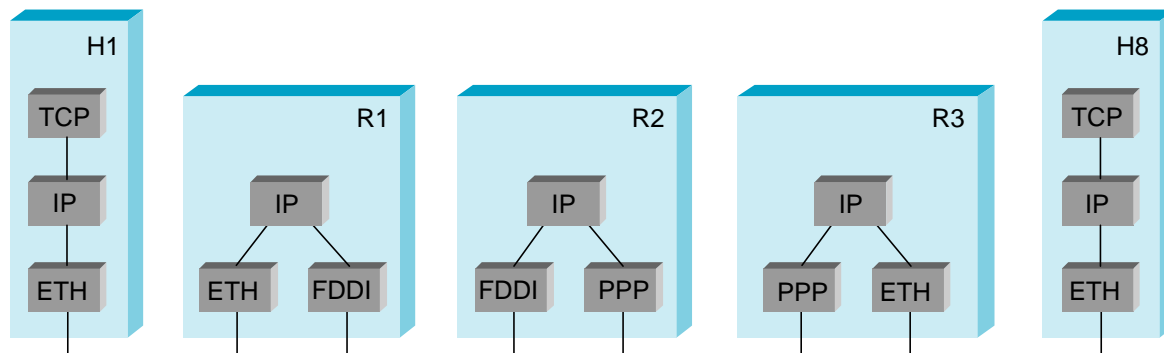
Global Addressing Scheme

# IP Internet

- Concatenation of Networks

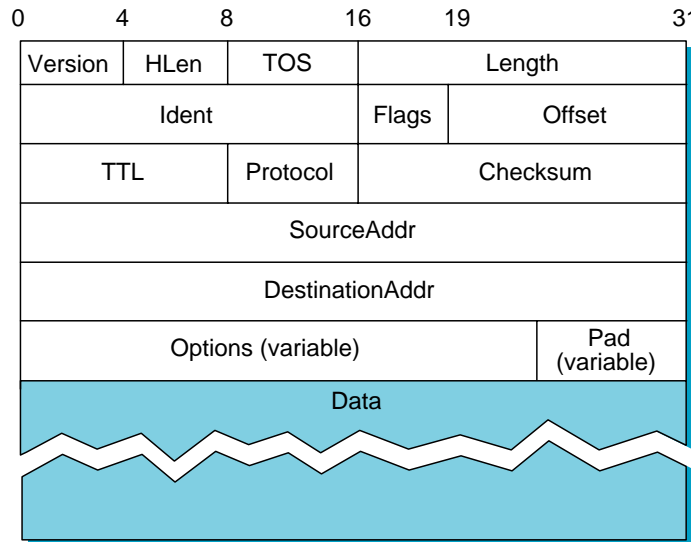


- Protocol Stack



# Service Model

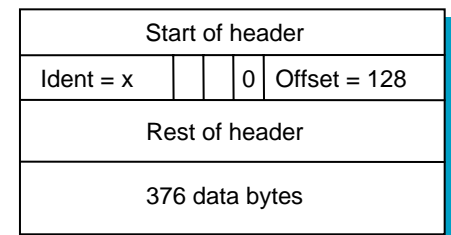
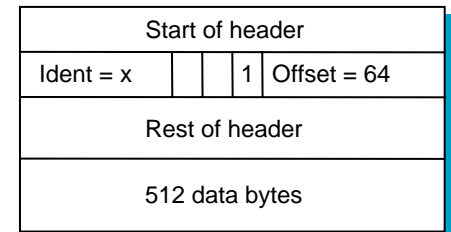
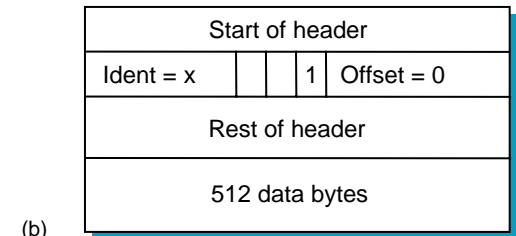
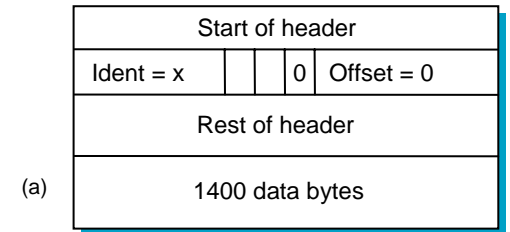
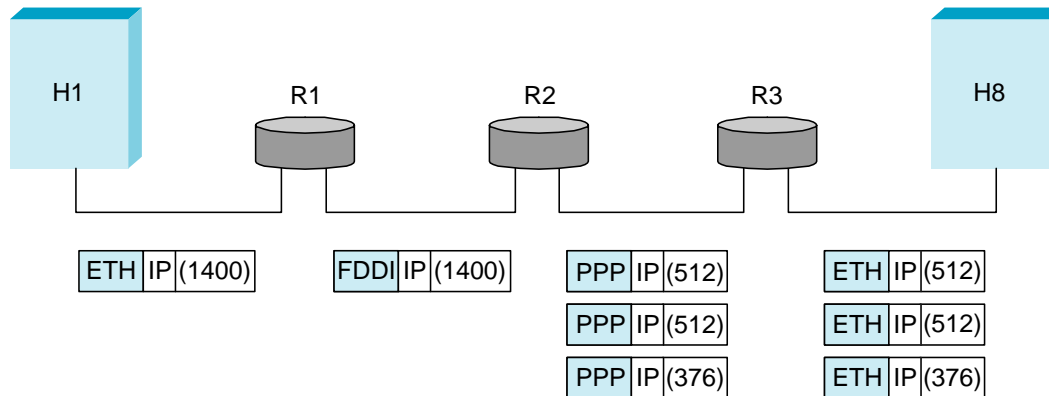
- Connectionless (datagram-based)
- Best-effort delivery (unreliable service)
  - packets are lost
  - packets are delivered out of order
  - duplicate copies of a packet are delivered
  - packets can be delayed for a long time
- Datagram format



# Fragmentation and Reassembly


- Each network has some MTU
- Design decisions
  - fragment when necessary ( $\text{MTU} < \text{Datagram}$ )
  - try to avoid fragmentation at source host
  - re-fragmentation is possible
  - fragments are self-contained datagrams
  - use CS-PDU (not cells) for ATM
  - delay reassembly until destination host
  - do not recover from lost fragments

# Example




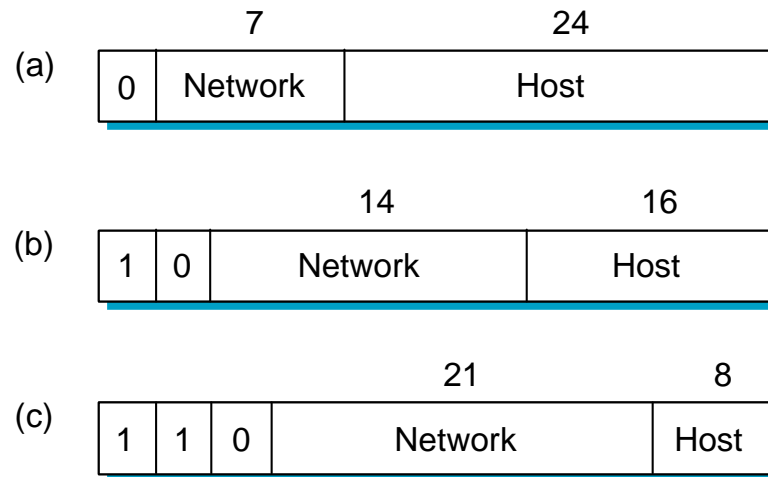
# Global Addresses

- Properties
    - globally unique
    - hierarchical: network + host
  - Dot Notation
    - 10.3.2.4
    - 128.96.33.81
    - 192.12.69.77
- (a)



(b)







# Datagram Forwarding

- Strategy
  - every datagram contains destination's address
  - if connected to destination network, then forward to host
  - if not directly connected, then forward to some router
  - forwarding table maps network number into next hop
  - each host has a default router
  - each router maintains a forwarding table

- Example (R2)

Network Number	Next Hop
1	R3
2	R1
3	interface 1
4	interface 0

# Address Translation

- Map IP addresses into physical addresses
  - destination host
  - next hop router
- Techniques
  - encode physical address in host part of IP address
  - table-based
- ARP
  - table of IP to physical address bindings
  - broadcast request if IP address not in table
  - target machine responds with its physical address
  - table entries are discarded if not refreshed

# ARP Details

- Request Format
  - HardwareType: type of physical network (e.g., Ethernet)
  - ProtocolType: type of higher layer protocol (e.g., IP)
  - HLEN & PLEN: length of physical and protocol addresses
  - Operation: request or response
  - Source/Target-Physical/Protocol addresses
- Notes
  - table entries timeout in about 10 minutes
  - update table with source when you are the target
  - update table if already have an entry
  - do not refresh table entries upon reference

# ARP Packet Format

0	8	16	31
Hardware type = 1		ProtocolType = 0x0800	
HLen = 48	PLen = 32	Operation	
SourceHardwareAddr (bytes 0 3)			
SourceHardwareAddr (bytes 4 5)		SourceProtocolAddr (bytes 0 1)	
SourceProtocolAddr (bytes 2 3)		TargetHardwareAddr (bytes 0 1)	
TargetHardwareAddr (bytes 2 5)			
TargetProtocolAddr (bytes 0 3)			

# Internet Control Message Protocol (ICMP)

- Echo (ping)
- Redirect (from router to source host)
- Destination unreachable (protocol, port, or host)
- TTL exceeded (so datagrams don't cycle forever)
- Checksum failed
- Reassembly failed
- Cannot fragment

# *Scalable Routing*

*Go To Talk Outline*

# Routing

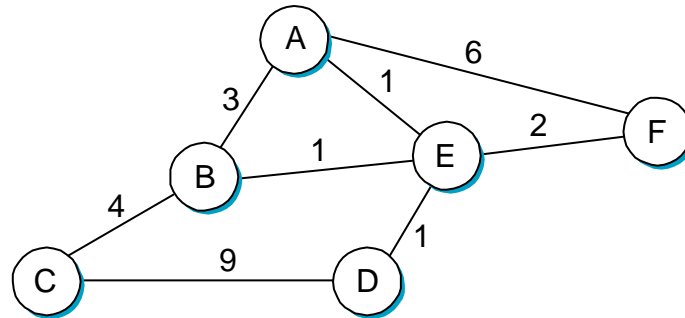
## Outline

Algorithms

Scalability

# Overview

- Forwarding vs Routing
  - forwarding: to select an output port based on destination address and routing table
  - routing: process by which routing table is built
- Network as a Graph



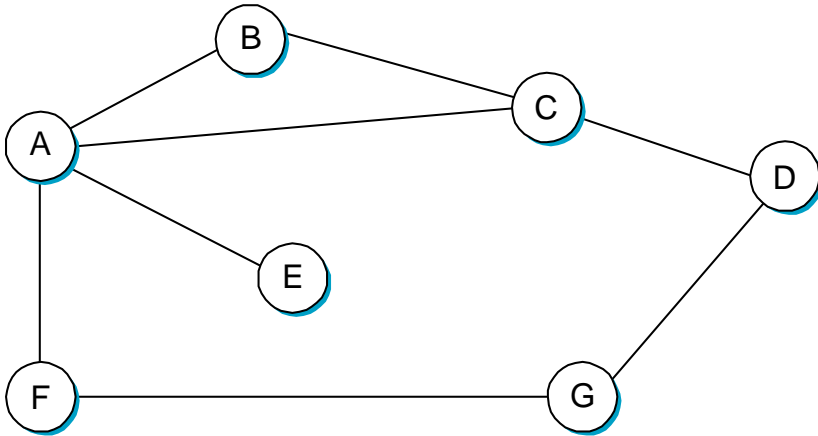
- Problem: Find lowest cost path between two nodes
- Factors
  - static: topology
  - dynamic: load



# Distance Vector

- Each node maintains a set of triples
  - (`Destination`, `Cost`, `NextHop`)
- Directly connected neighbors exchange updates
  - periodically (on the order of several seconds)
  - whenever table changes (called *triggered* update)
- Each update is a list of pairs:
  - (`Destination`, `Cost`)
- Update local table if receive a “better” route
  - smaller cost
  - came from next-hop
- Refresh existing routes; delete if they time out

# Example



Destination	Cost	NextHop
A	1	A
C	1	C
D	2	C
E	2	A
F	2	A
G	3	A

# Routing Loops

- Example 1
  - F detects that link to G has failed
  - F sets distance to G to infinity and sends update to A
  - A sets distance to G to infinity since it uses F to reach G
  - A receives periodic update from C with 2-hop path to G
  - A sets distance to G to 3 and sends update to F
  - F decides it can reach G in 4 hops via A
- Example 2
  - link from A to E fails
  - A advertises distance of infinity to E
  - B and C advertise a distance of 2 to E
  - B decides it can reach E in 3 hops; advertises this to A
  - A decides it can reach E in 4 hops; advertises this to C
  - C decides that it can reach E in 5 hops...

# Loop-Breaking Heuristics

- Set infinity to 16
- Split horizon
- Split horizon with poison reverse

# Link State

- Strategy
  - send to all nodes (not just neighbors)  
information about directly connected links (not entire routing table)
- Link State Packet (LSP)
  - id of the node that created the LSP
  - cost of link to each directly connected neighbor
  - sequence number (SEQNO)
  - time-to-live (TTL) for this packet

## Link State (cont)

- Reliable flooding
  - store most recent LSP from each node
  - forward LSP to all nodes but one that sent it
  - generate new LSP periodically
    - increment SEQNO
  - start SEQNO at 0 when reboot
  - decrement TTL of each stored LSP
    - discard when TTL=0

# Route Calculation

- Dijkstra's shortest path algorithm
- Let
  - $N$  denotes set of nodes in the graph
  - $l(i, j)$  denotes non-negative cost (weight) for edge  $(i, j)$
  - $s$  denotes this node
  - $M$  denotes the set of nodes incorporated so far
  - $C(n)$  denotes cost of the path from  $s$  to node  $n$

$M = \{s\}$

for each  $n$  in  $N - \{s\}$

$C(n) = l(s, n)$

while ( $N \neq M$ )

$M = M \text{ union } \{w\} \text{ such that } C(w) \text{ is the minimum for}$   
 $\text{all } w \text{ in } (N - M)$

for each  $n$  in  $(N - M)$

$C(n) = \text{MIN}(C(n), C(w) + l(w, n))$

# Metrics

- Original ARPANET metric
  - measures number of packets queued on each link
  - took neither latency or bandwidth into consideration
- New ARPANET metric
  - stamp each incoming packet with its arrival time (**AT**)
  - record departure time (**DT**)
  - when link-level ACK arrives, compute
$$\text{Delay} = (\text{DT} - \text{AT}) + \text{Transmit} + \text{Latency}$$
  - if timeout, reset **DT** to departure time for retransmission
  - link cost = average delay over some time period
- Fine Tuning
  - compressed dynamic range
  - replaced **Delay** with link utilization

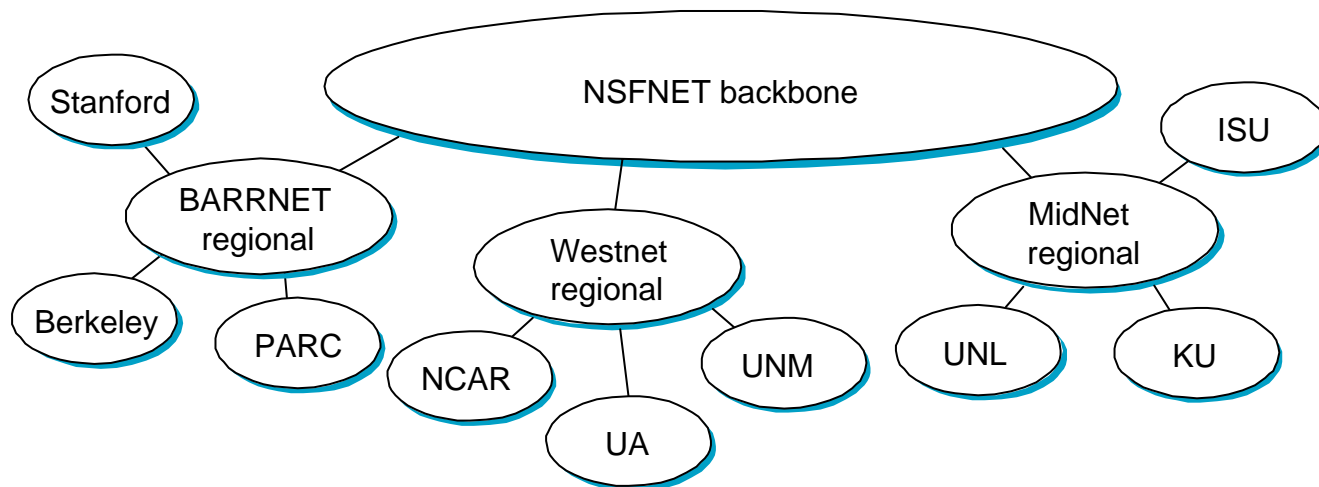


# How to Make Routing Scale

- Flat versus Hierarchical Addresses
- Inefficient use of Hierarchical Address Space
  - class C with 2 hosts ( $2/255 = 0.78\%$  efficient)
  - class B with 256 hosts ( $256/65535 = 0.39\%$  efficient)
- Still Too Many Networks
  - routing tables do not scale
  - route propagation protocols do not scale

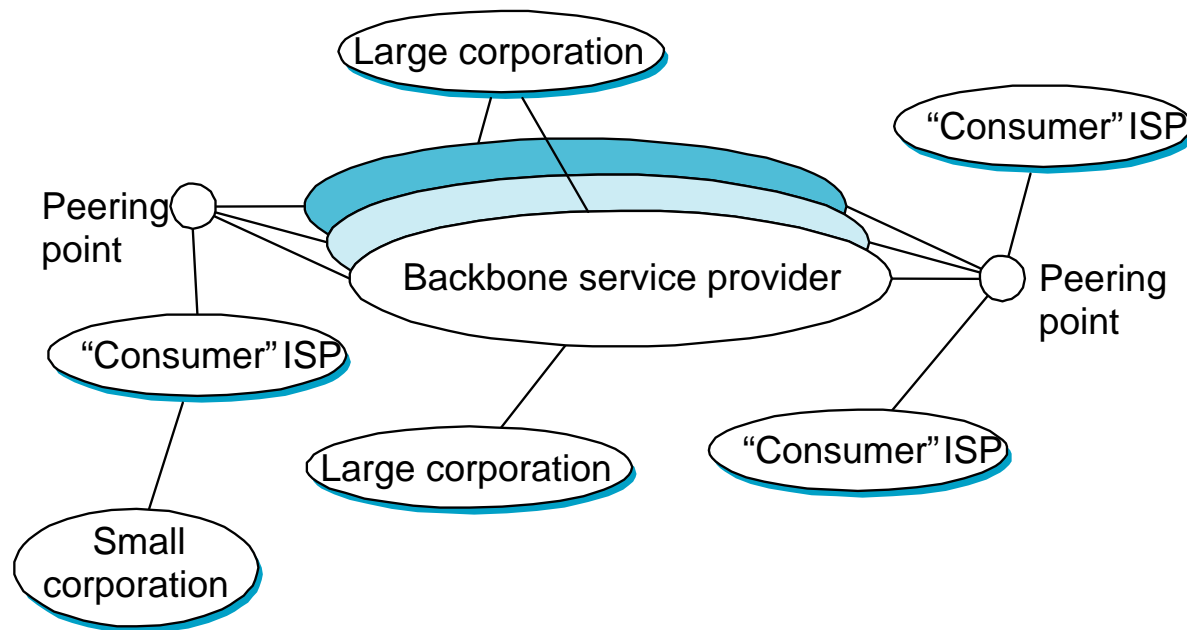
# Internet Structure

## Recent Past



# Internet Structure

Today

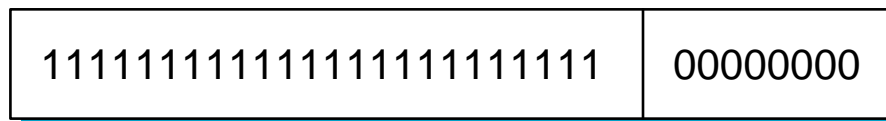


# Subnetting

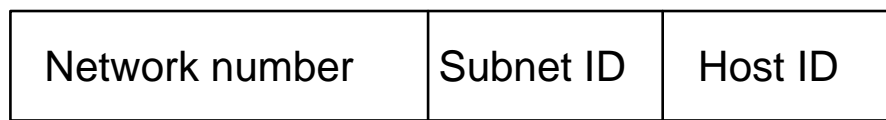
- Add another level to address/routing hierarchy: *subnet*
- *Subnet masks* define variable partition of host part
- Subnets visible only within site



Class B address

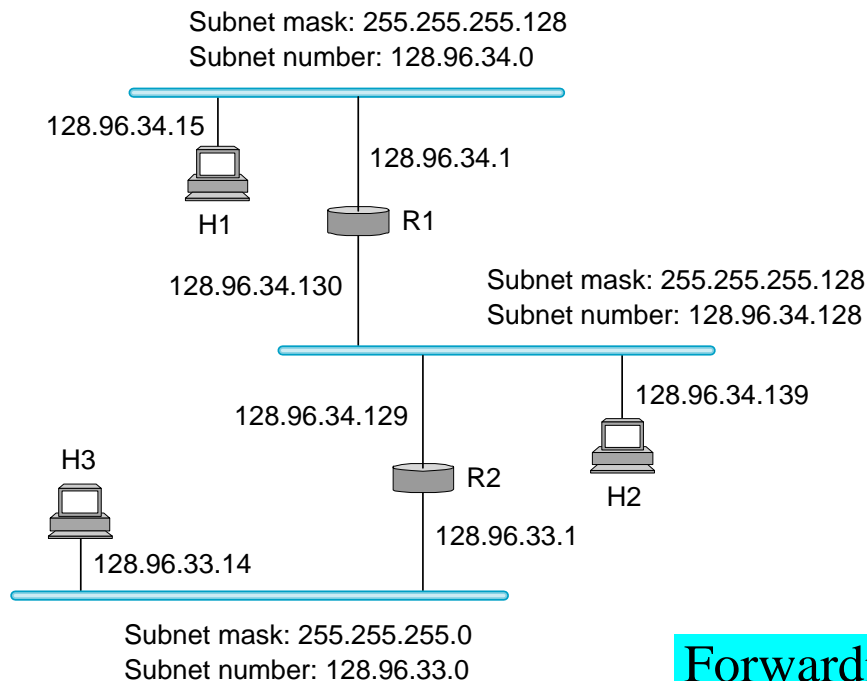


Subnet mask (255.255.255.0)



Subnetted address

# Subnet Example



## Forwarding table at router R1

Subnet Number	Subnet Mask	Next Hop
128.96.34.0	255.255.255.128	interface 0
128.96.34.128	255.255.255.128	interface 1
128.96.33.0	255.255.255.0	R2

# Forwarding Algorithm

```
D = destination IP address
for each entry (SubnetNum, SubnetMask, NextHop)
    D1 = SubnetMask & D
    if D1 = SubnetNum
        if NextHop is an interface
            deliver datagram directly to D
        else
            deliver datagram to NextHop
```

- Use a default router if nothing matches
- Not necessary for all 1s in subnet mask to be contiguous
- Can put multiple subnets on one physical network
- Subnets not visible from the rest of the Internet

# Supernetting

- Assign block of contiguous network numbers to nearby networks
- Called CIDR: Classless Inter-Domain Routing
- Represent blocks with a single pair  
**(first\_network\_address, count)**
- Restrict block sizes to powers of 2
- Use a bit mask (CIDR mask) to identify block size
- All routers must understand CIDR addressing

# Route Propagation

- Know a smarter router
  - hosts know local router
  - local routers know site routers
  - site routers know core router
  - core routers know everything
- Autonomous System (AS)
  - corresponds to an administrative domain
  - examples: University, company, backbone network
  - assign each AS a 16-bit number
- Two-level route propagation hierarchy
  - interior gateway protocol (each AS selects its own)
  - exterior gateway protocol (Internet-wide standard)



# Popular Interior Gateway Protocols

- RIP: Route Information Protocol
  - developed for XNS
  - distributed with Unix
  - distance-vector algorithm
  - based on hop-count
- OSPF: Open Shortest Path First
  - recent Internet standard
  - uses link-state algorithm
  - supports load balancing
  - supports authentication

# EGP: Exterior Gateway Protocol

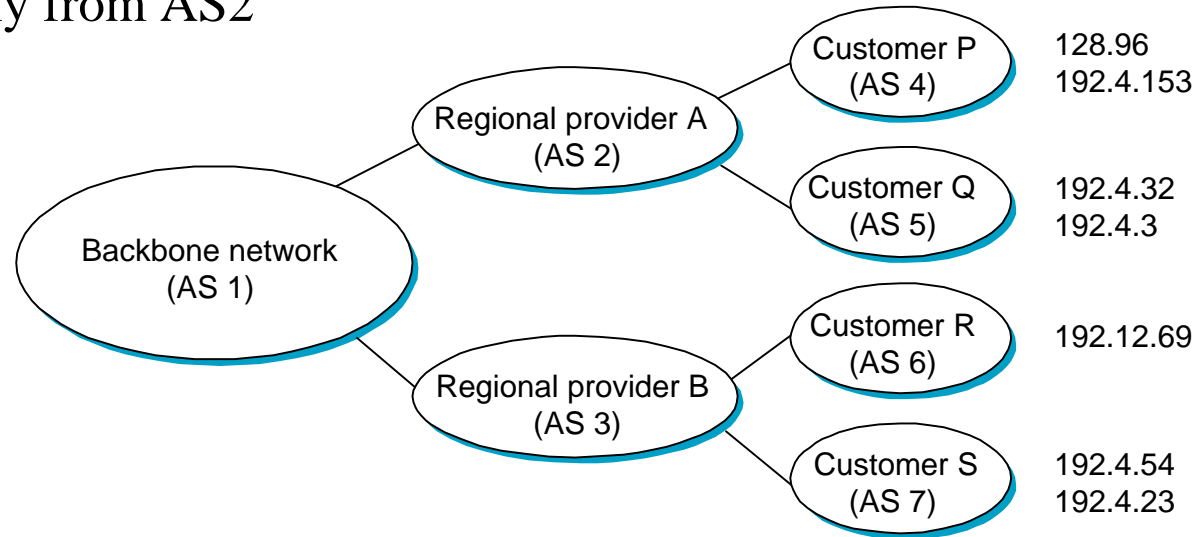
- Overview
  - designed for tree-structured Internet
  - concerned with *reachability*, not optimal routes
- Protocol messages
  - neighbor acquisition: one router requests that another be its peer; peers exchange reachability information
  - neighbor reachability: one router periodically tests if the another is still reachable; exchange HELLO/ACK messages; uses a k-out-of-n rule
  - routing updates: peers periodically exchange their routing tables (distance-vector)

# BGP-4: Border Gateway Protocol

- AS Types
  - stub AS: has a single connection to one other AS
    - carries local traffic only
  - multihomed AS: has connections to more than one AS
    - refuses to carry transit traffic
  - transit AS: has connections to more than one AS
    - carries both transit and local traffic
- Each AS has:
  - one or more border routers
  - one BGP *speaker* that advertises:
    - local networks
    - other reachable networks (transit AS only)
    - gives *path* information

# BGP Example

- Speaker for AS2 advertises reachability to P and Q
  - network 128.96, 192.4.153, 192.4.32, and 192.4.3, can be reached directly from AS2



- Speaker for backbone advertises
  - networks 128.96, 192.4.153, 192.4.32, and 192.4.3 can be reached along the path (AS1, AS2).
- Speaker can cancel previously advertised paths

# IP Version 6

- Features
  - 128-bit addresses (classless)
  - multicast
  - real-time service
  - authentication and security
  - autoconfiguration
  - end-to-end fragmentation
  - protocol extensions
- Header
  - 40-byte “base” header
  - extension headers (fixed order, mostly fixed length)
    - fragmentation
    - source routing
    - authentication and security
    - other options

# Group Communication

## Outline

Multicast Routing

Logical Time

Order & Membership Protocols

# Process Groups

- Any set of processes that want to cooperate
- Processes can join/leave either implicitly or explicitly
- A process can belong to many groups
- Groups can be either open or closed
- Use multicast rather than point-to-point messages
  - group name (address) provides a useful level of indirection
- Example uses
  - data dissemination (e.g., news)
  - replicated servers

# Multicast Routing: LS

- Each host on a LAN periodically announces the groups it belongs to using IGMP
- Augment update message (LSP) to include set of groups that have members on a particular LAN
- Each router uses Dijkstra's algorithm to compute shortest-path spanning tree for each source/group pair
- Each router caches tree for currently active source/group pairs



# Multicast Routing: DV

- Reverse Path Broadcast
  - Each router already knows that shortest path to S goes through router N
  - When receive multicast packet from S, forward on all outgoing links (except one it arrived on), iff packet arrived from N
  - Eliminate duplicate broadcast packets by letting only “parent” for LAN (relative to S) forward
    - shortest path to S (learn from distance vector)
    - smallest address to break ties

## DV (cont)

- Reverse Path Multicast
  - Goal: prune networks that have no hosts in group G
  - Step 1: determine if LAN is a *leaf* w/ no members in G
    - leaf if parent is only router on the LAN
    - determine if any hosts are members of G using IGMP
  - Step 2: propagate “no members of G here” information
    - augment (destination, cost) update sent to neighbors with set of groups for which this network is interested in receiving multicast packets
    - only happens when multicast address becomes active

# Replicated State Machine

- Service is characterized as a state machine that modifies variables in response to outside operations
- State machine is replicated to improve availability
- Key is ensuring
  - all operations are atomic (applied at all functioning replicas)
  - all replicas remain consistent (ops applied in same order)
- Implementation
  - encapsulate operations in messages
  - send using group communication

# Atomic Messages

- Atomicity property: a message is delivered to all members, or to none
- First try...
  - each recipient acknowledges message
  - sender retransmits if ACK not received
  - problem: sender could crash before message is delivered everywhere

## Atomic Messages (cont)

- Fix: if sender crashes, a recipient volunteers to be “backup sender” for the message
  - re-sends message to everybody, waits for ACKs
  - use simple algorithm to choose volunteer
  - apply method again if backup fails
- Must remember all received messages in case we need to become backup sender
  - periodic protocol to “prune” old messages
  - how know it’s safe to prune?

# Message Ordering

- So far: different members may see messages in different orders
- Ordered group communication requires all members to agree about the order of messages
- Within group, assign global ordering to messages
- Hold back messages that arrive out-of-order

# Ordering: First Approach

- Central ordering server assigns global sequence numbers
- Hosts apply to ordering server for numbers, or ordering server sends all messages itself
- Have to deal with case where ordering server fails
  - leader election we saw earlier
- Hold-back easy since sequence numbers are sequential

# Ordering: Second Approach

- Use time message was sent
  - measured on sending host
  - use host address to break ties
- Advantage
  - simple and decentralized
- Disadvantage
  - requires nearly synchronized clocks
  - must hold back messages for a period equal to maximum clock difference

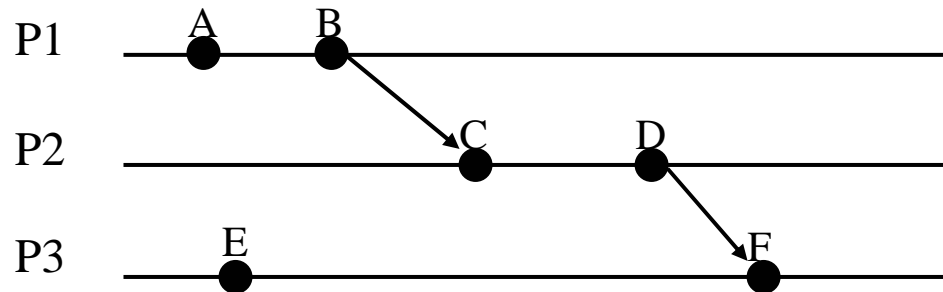


# Logical Time

- Insight: often don't care about when something happened, only about which thing happened first
- Happened before relationship
  - $X < Y$  means “X happened before Y”
  - three rules:
    - if X and Y occur in the same process and X occurs before Y, then  $X < Y$
    - if M is a message, then  $\text{send}(M) < \text{receive}(M)$
    - if  $X < Y$  and  $Y < Z$ , then  $X < Z$

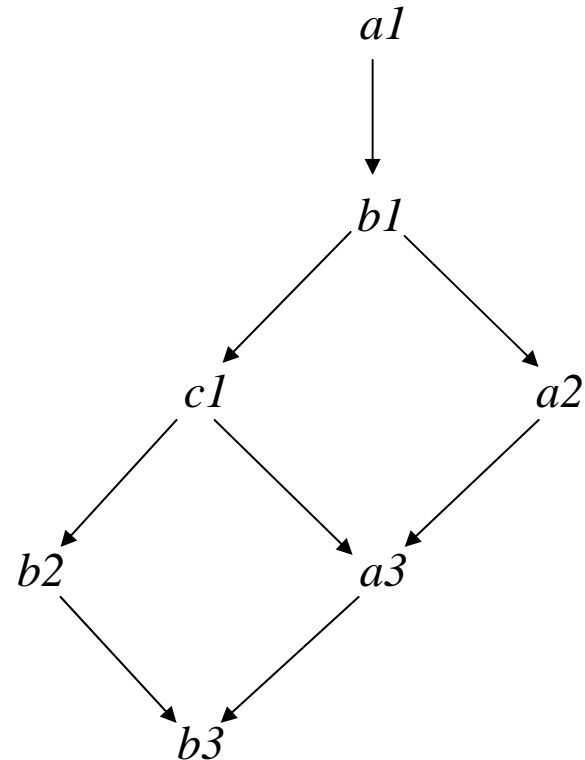
# Logical Time (cont)

- Given two events  $X$  and  $Y$ , either
  - $X < Y$ , or
  - $Y < X$ , or
  - neither ( $X$  and  $Y$  are concurrent)
- $<$  relation defines a partial order
- Example



# Message Context

- A process sends a message *in the context* of all the messages it has received.
- Group communication represented with a *context graph*.
- Example: 3 senders, denoted  $a$ ,  $b$ , and  $c$



# Protocol

- Each server maintains a copy of the context graph
  - union of all copies equals “global graph”
- Send: mid + mid of all predecessor messages
  - leaves of sender’s copy of context graph
  - bounded by number of participants (why?)
- Receive: add to local copy and deliver to application
  - hold back if not all predecessors are present
  - ask sender to retransmit missing messages (why?)
  - pass up to application in “context” order

# Protocol (cont)

- Applications can inspect context graph
  - leaves, precedes, prev, root, stable
- Message stability
  - followed by a message from all other participants
- System can free all stable messages from its copy
  - will never be asked to retransmit them

# Host Failures

- Guarantees
  - all running processes are able to continue exchanging messages
  - a message contained in any running host's copy will eventually be incorporated into every running host's copy
- Application support
  - mask out failed processes
  - adjusts message stability

# Message Order

- Context graph preserves partial order among messages
- Each host can produce same total order by running a topological sort on context graph
  - incremental since messages continually arriving
- Commit next “wave” of messages to application as soon as one message in wave becomes stable
  - know that no future messages will be at same logical time
- Membership protocol much trickier

# Router Construction

## Outline

Switched Fabrics

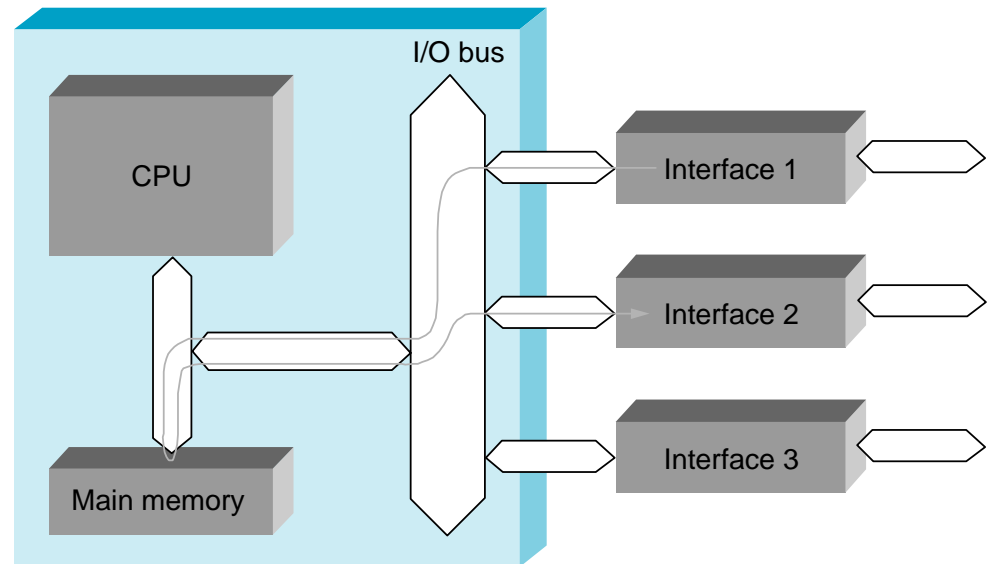
IP Routers

Tag Switching



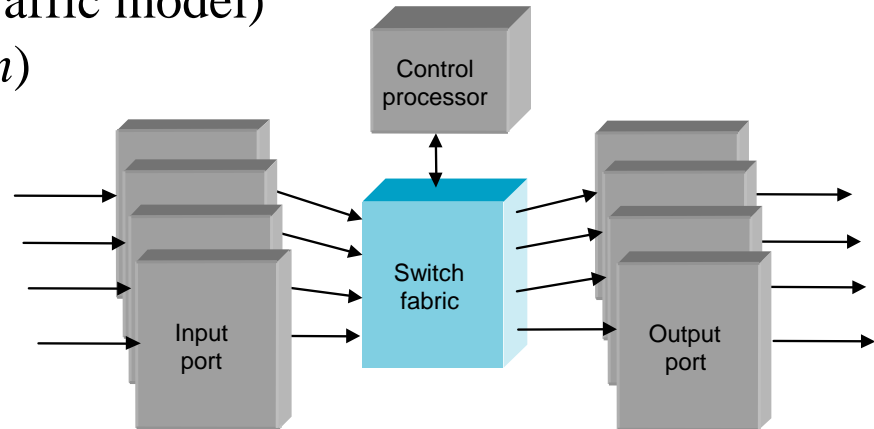
# Workstation-Based

- Aggregate bandwidth
  - 1/2 of the I/O bus bandwidth
  - capacity shared among all hosts connected to switch
  - example: 1Gbps bus can support 5 x 100Mbps ports (in theory)
- Packets-per-second
  - must be able to switch small packets
  - 300,000 packets-per-second is achievable
  - e.g., 64-byte packets implies 155Mbps



# Switching Hardware

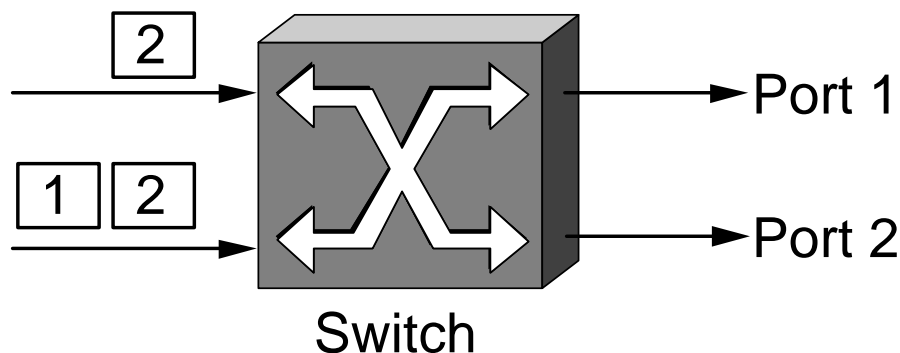
- Design Goals
  - throughput (depends on traffic model)
  - scalability (a function of  $n$ )



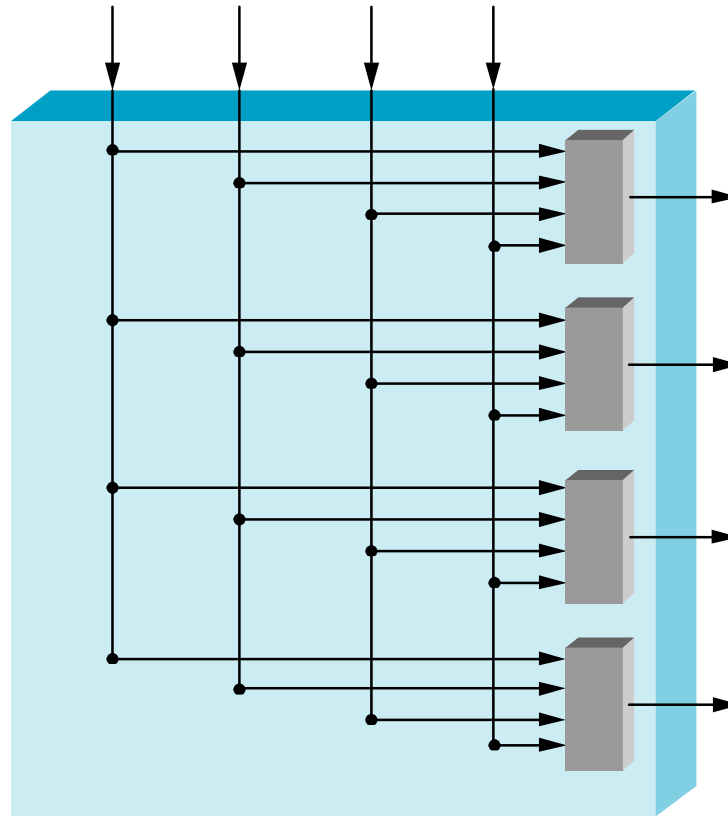
- Ports
  - circuit management (e.g., map VCIs, route datagrams)
  - buffering (input and/or output)
- Fabric
  - as simple as possible
  - sometimes do buffering (internal)

# Buffering

- Wherever contention is possible
  - input port (contend for fabric)
  - internal (contend for output port)
  - output port (contend for link)
- Head-of-Line Blocking
  - input buffering

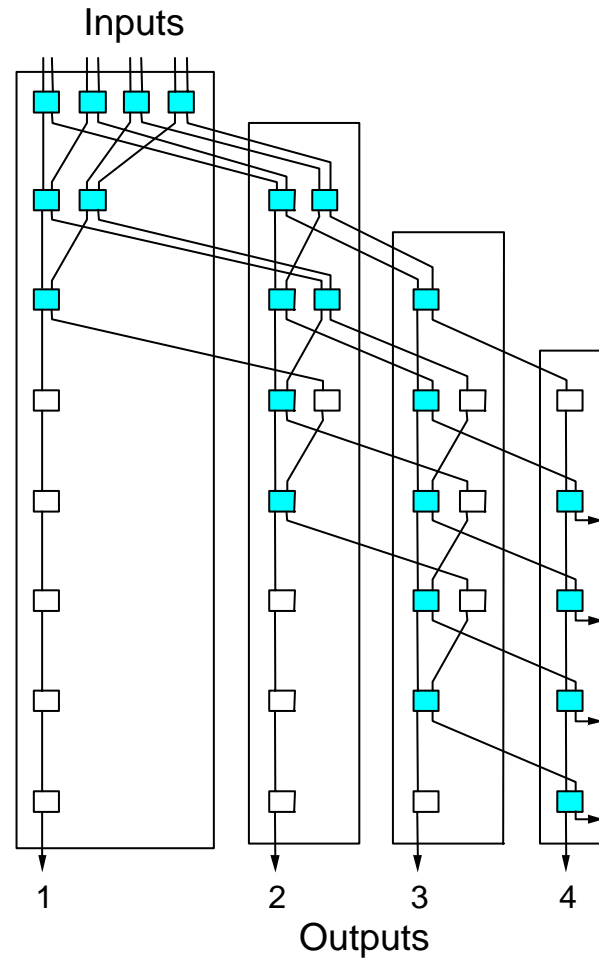


# Crossbar Switches



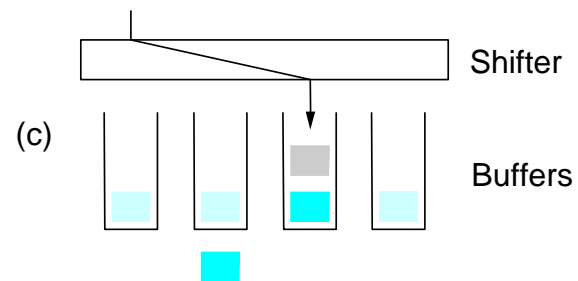
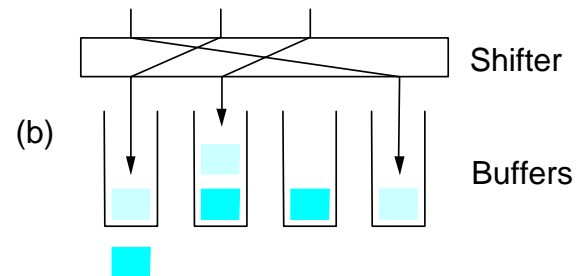
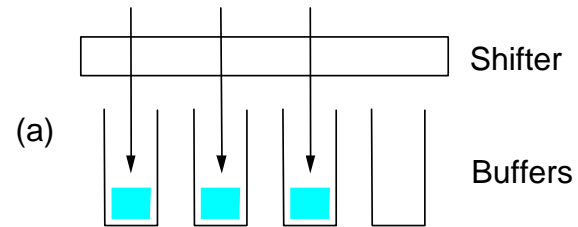
# Knockout Switch

- Example crossbar
- Concentrator
  - select  $l$  of  $n$  packets
- Complexity:  $n^2$



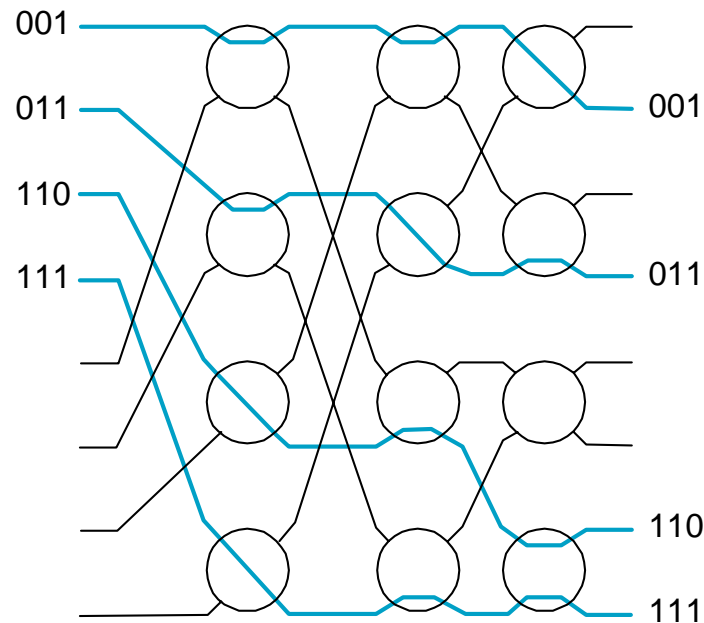
# Knockout Switch (cont)

- Output Buffer



# Self-Routing Fabrics

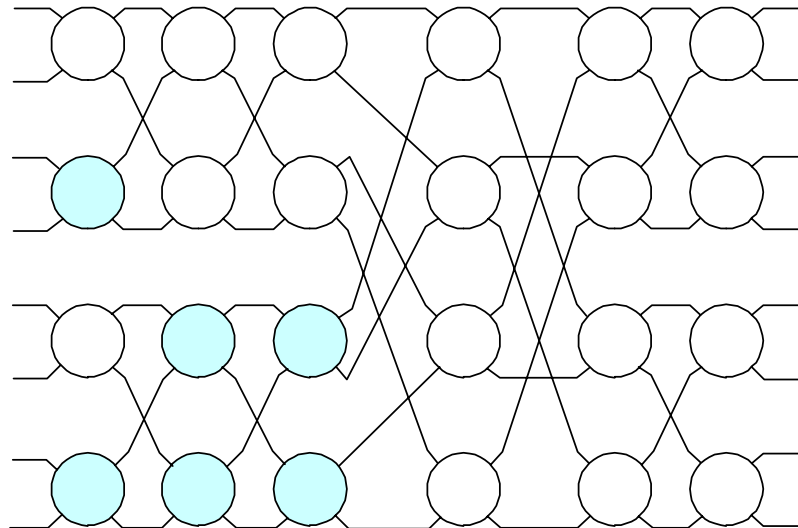
- Banyan Network
  - constructed from simple  $2 \times 2$  switching elements
  - self-routing header attached to each packet
  - elements arranged to route based on this header
  - no collisions if input packets sorted into ascending order
  - complexity:  $n \log_2 n$



# Self-Routing Fabrics (cont)

- Batcher Network

- switching elements sort two numbers
  - some elements sort into ascending (clear)
  - some elements sort into descending (shaded)
- elements arranged to implement merge sort
- complexity:  $n \log^2_2 n$

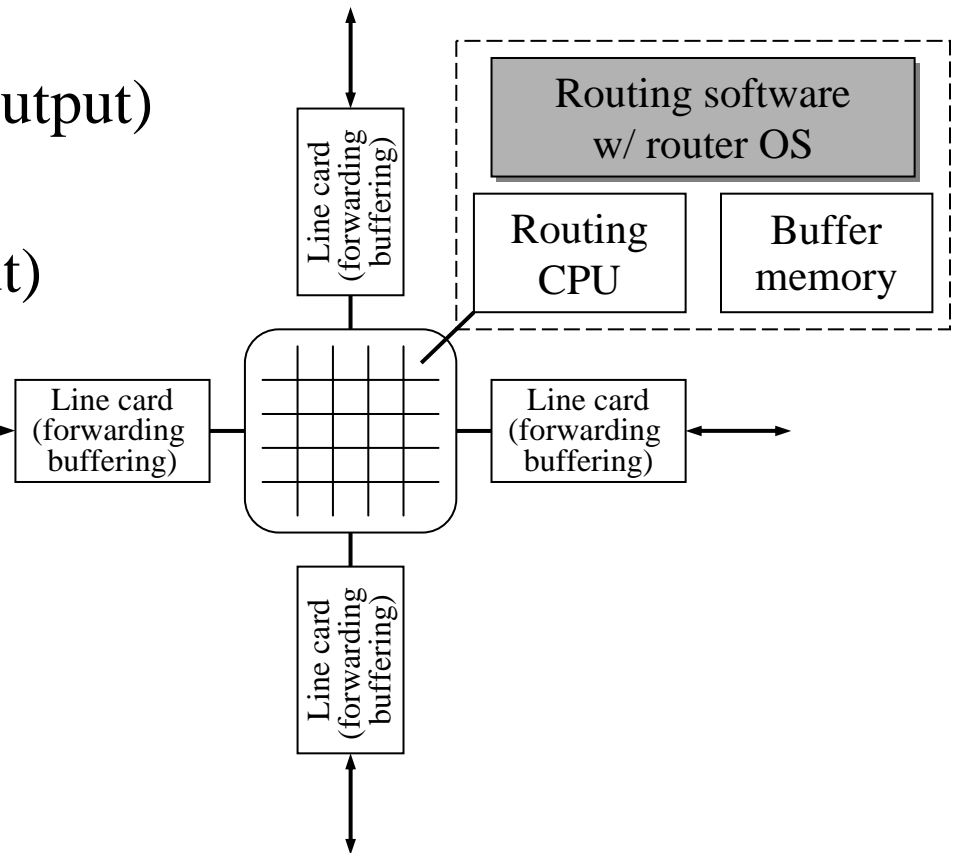


- Common Design: Batcher-Banyan Switch



# High-Speed IP Router

- Switch (possibly ATM)
- Line Cards
  - link interface (input, output)
  - router lookup (input)
  - common IP path (input)
  - packet queue (output)
- Control Processor
  - routing protocol(s)
  - exceptional cases



# IP Forwarding is Slow

- Problem: classless IP addresses (CIDR)
- Route by variable-length Forwarding Equivalence Classes (FEC)
  - FEC = IP address plus prefix of 1-32 bits; e.g., 172.200.0.0/16
- IP Router
  - forwarding tbl:  $\langle \text{FEC} \rangle \longrightarrow \langle \text{next hop, port} \rangle$
  - match IP address to FEC w/ longest prefix

# ATM Forwarding

- Primary goal: fast, cheap forwarding
- 1Gb/s IP router: \$187,000
- 5Gb/s ATM switch: \$41,000
- Create Virtual Circuit at Flow Setup
  - $\langle \text{in VCI} \rangle \longrightarrow \langle \text{port, out VCI} \rangle$
- Cell Forwarding
  - index, swap, switch

# Cisco: Tag Switching

- Add a VCI-like tag to packets
  - $\langle \text{in tag} \rangle \longrightarrow \langle \text{next hop, port, out tag} \rangle$
- TSR uses ATM switch hardware
- IP routing protocols (OSPF, RIP, BGP)
  - build forwarding table from routing table
- Goal: IP router functionality at ATM switch speeds/costs

# Forwarding

- *Shim* before IP header



- Tag Forwarding Information Base (TFIB)
  - $\langle \text{in tag} \rangle \longrightarrow \langle \text{next hop, port, out tag} \rangle$
- Just like ATM
  - index, swap, switch

# Tag Binding

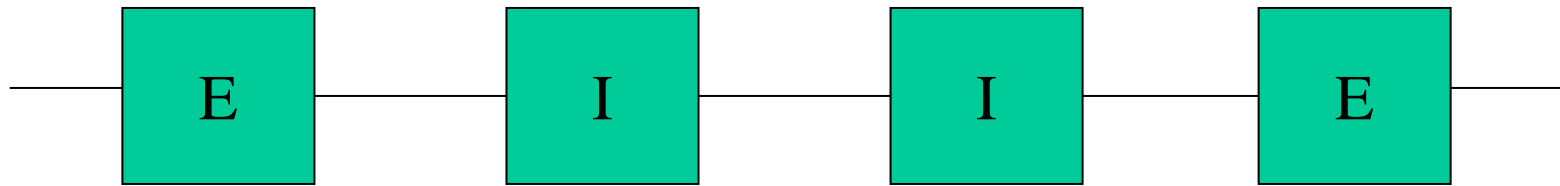
- New FEC from IP routing protocols
  - Select local tag (index in TFIB)
  - $\langle \text{in tag} \rangle \longrightarrow \langle \text{next hop, port, ???} \rangle$
- Need  $\langle \text{out tag} \rangle$  for next hop
- Other routers need my  $\langle \text{in tag} \rangle$
- Solution: distribute tags like other routing info

# Tag Distribution Protocol

- Send TDP messages to peers
  - $\langle \text{FEC}, \text{my tag} \rangle$
- Upon receiving TDP message, check if sender is next hop for FEC
  - yes, save tag in TFIB
  - no, can discard or save for future use
- ‘Control-driven’ label assignment

# The First Tag

- Two kinds of routers: edge vs. interior



- Edge: add shim based on IP lookup, strip at exit
- Interior: forward by tag only



# Robustness Issues

- What if tag fault?
  - try to forward (default route)
  - discard packet
- Forwarding Loops
  - topology changes cause temporary loops
  - TTL field in tag, same as IP

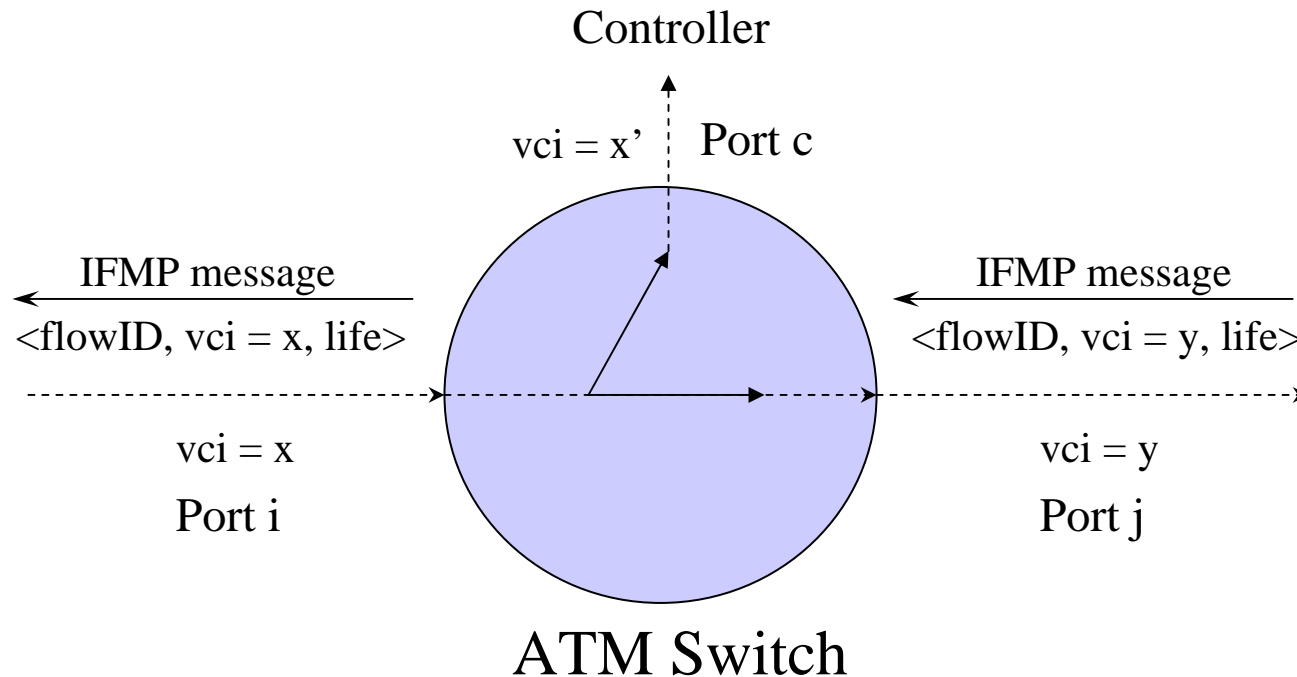
# Ipsilon: IP Switching

- Run on ATM switch over ATM network
  - ATM hardware + IP switching software
- Idea: Exploit temporal locality of traffic to cache routing decisions
- Associate labels (VCI) with flows
  - forward packets as usual
  - main difference is in how labels are created, distributed to other routers

# IP Switch

- Assume default ATM virtual circuits between routers
- Router runs IP routing protocol, can forward IP packets on default VCs
- Identify flows, assign flow-specific VC
  - flow = port pair or host pair
- ‘Data-driven’ label assignment

# Flow Setup on IP Switch



- $\langle \text{vci} = x \rangle \longrightarrow \langle \text{port c, vci} = x' \rangle$
- Get IFMP,  $\langle \text{vci} = x \rangle \longrightarrow \langle \text{port j, vci} = y \rangle$

# Comparison

## IP Switching

- Switch by flow
- Data driven
- Soft-state timeout
- Between end-hosts
- Every router can do IP lookup
- Scalable?

## Tag Switching

- Switch by FEC
- Control driven
- Route changes
- Between edge TSRs
- Interior TSRs only do tag switching

# Router Construction II

## Outline

- Network Processors

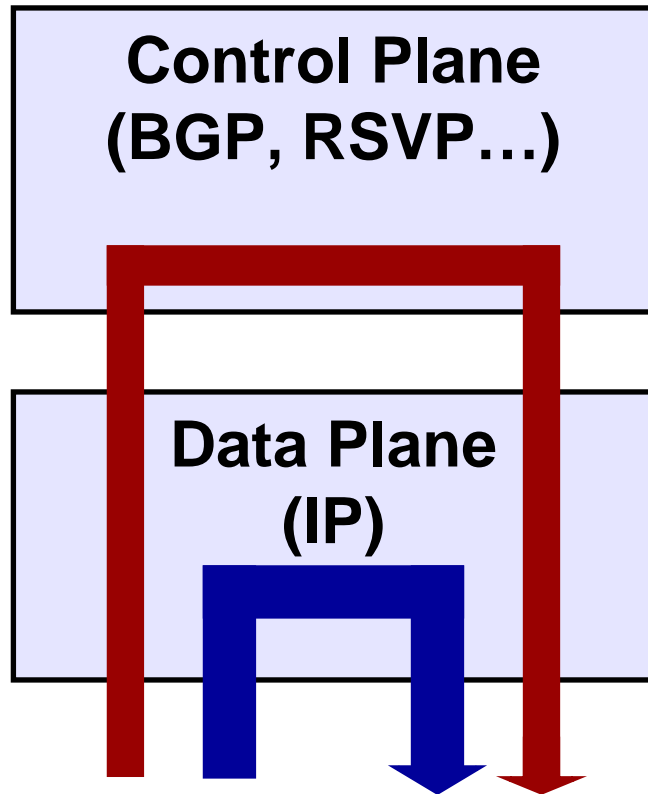
- Adding Extensions

- Scheduling Cycles

# Observations

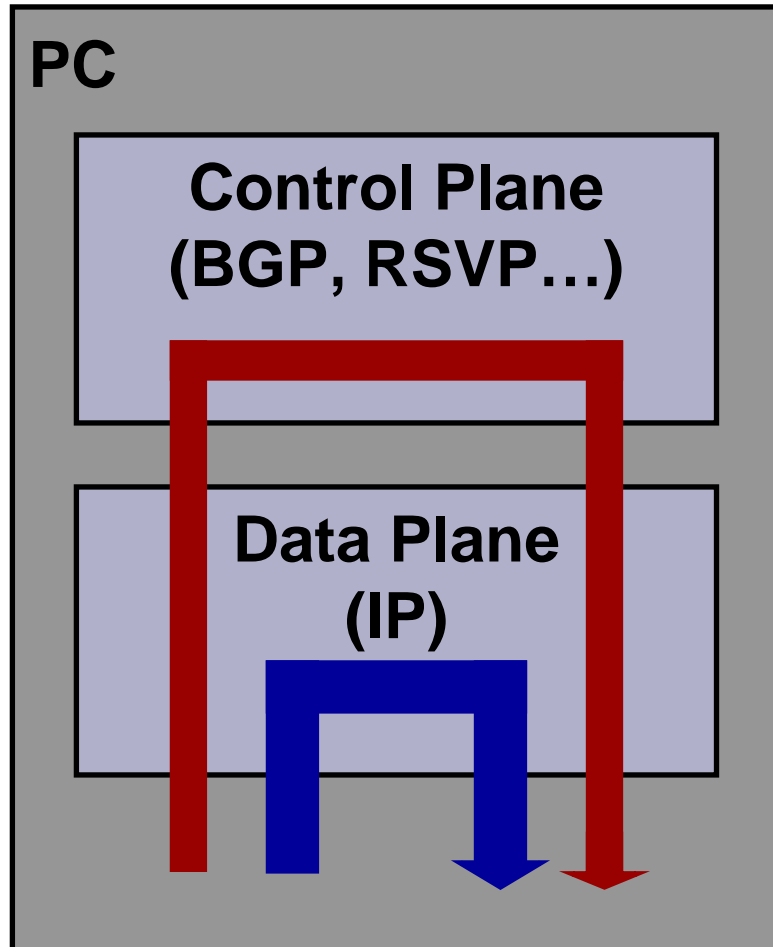
- Emerging commodity components can be used to build IP routers
  - switching fabrics, network processors, ...
- Routers are being asked to support a growing array of services
  - firewalls, proxies, p2p nets, overlays, ...

# Router Architecture



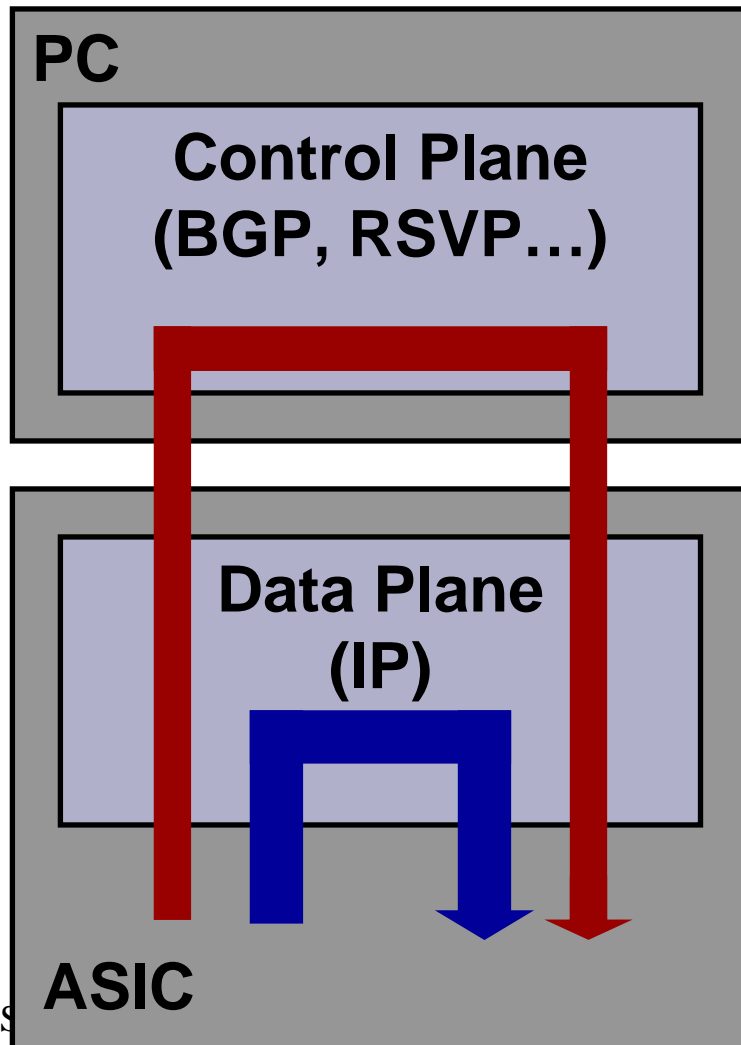


# Software-Based Router



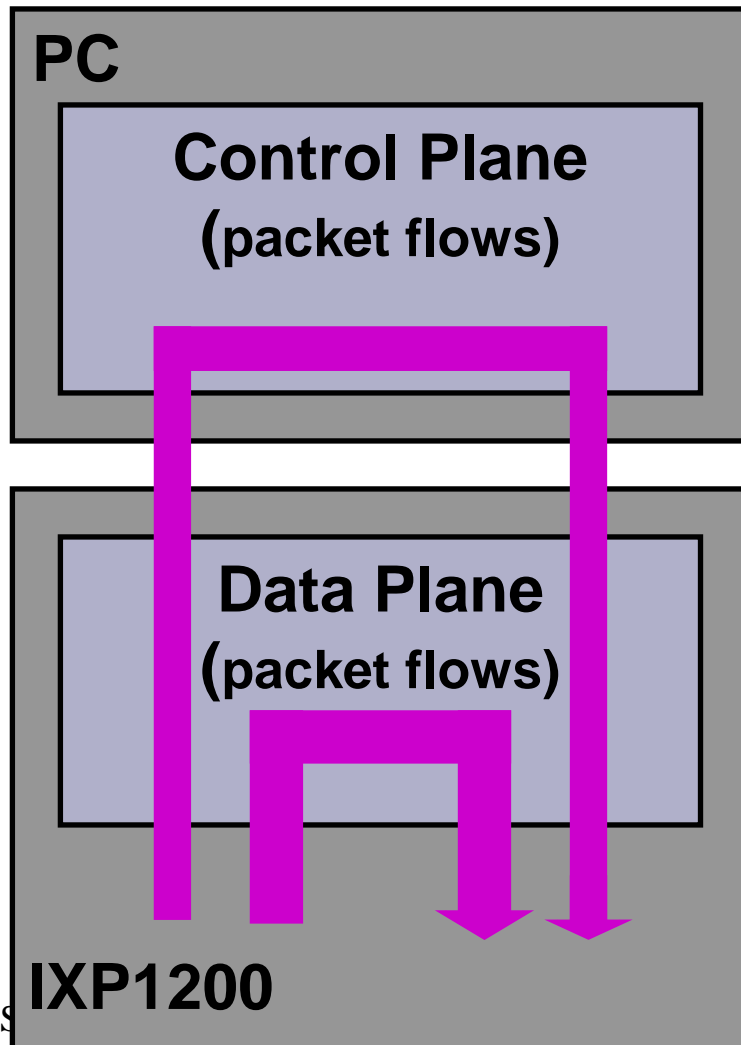
- + Cost
- + Programmability
- Performance (~300 Kpps)
- Robustness

# Hardware-Based Router



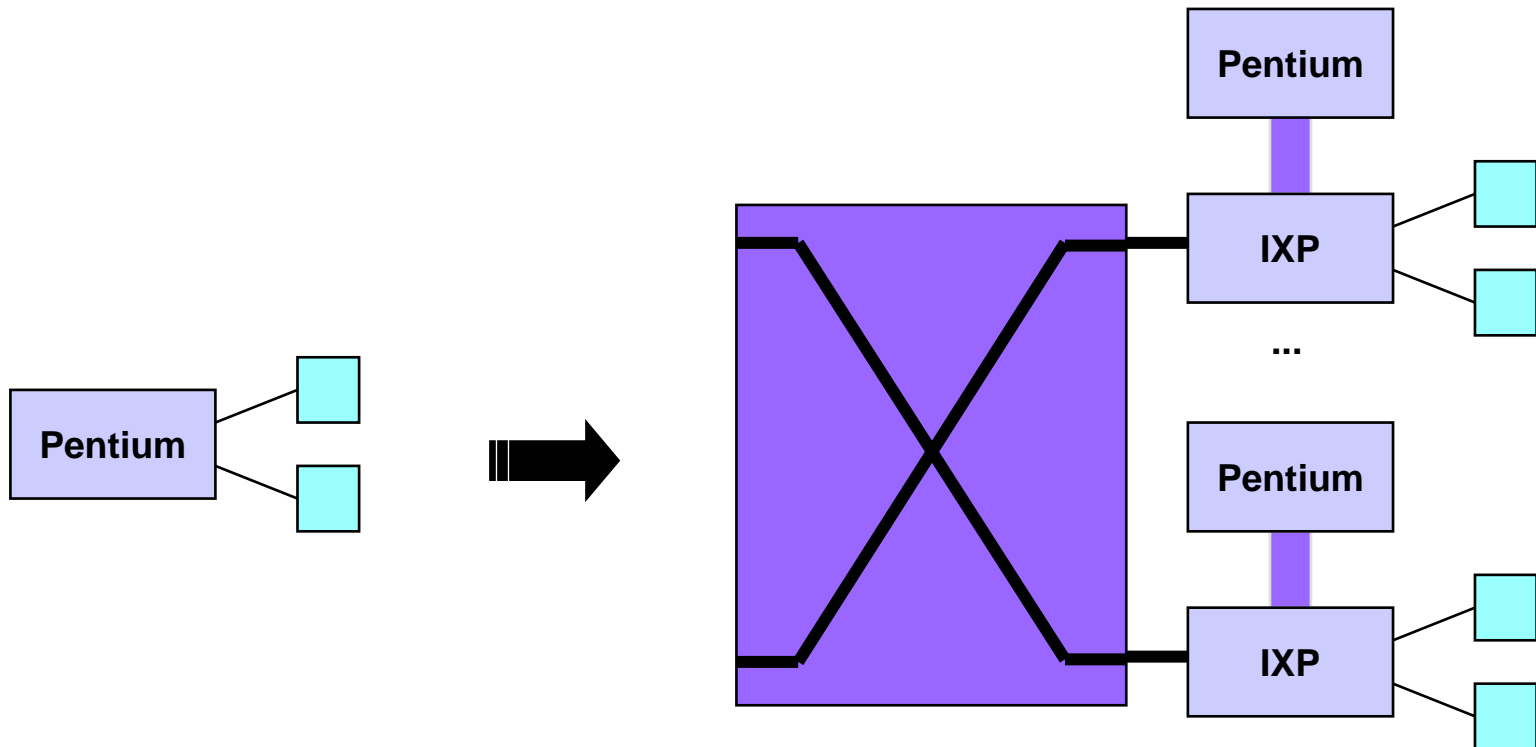
- Cost
- Programmability
- + Performance (25+ Mpps)
- + Robustness

# NP-Based Router Architecture

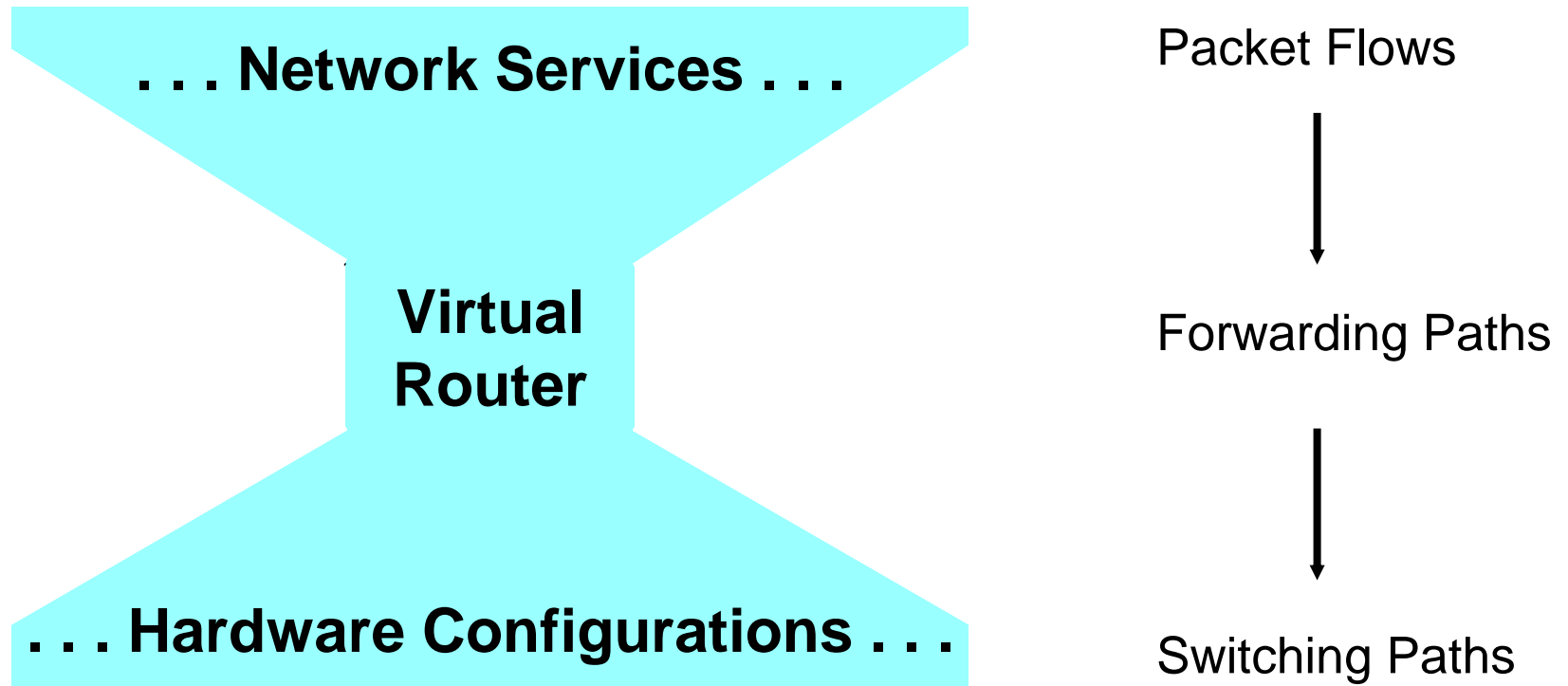


- + Cost (\$1500)
- + Programmability
- ? Performance
- ? Robustness

# In General...

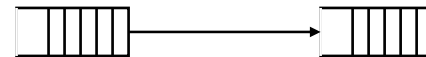
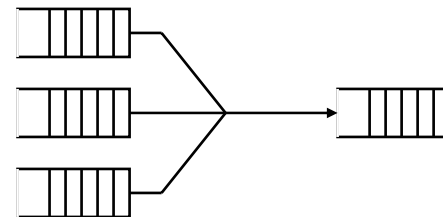
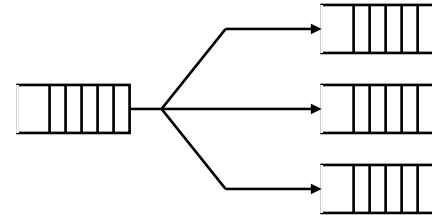


# Architectural Overview

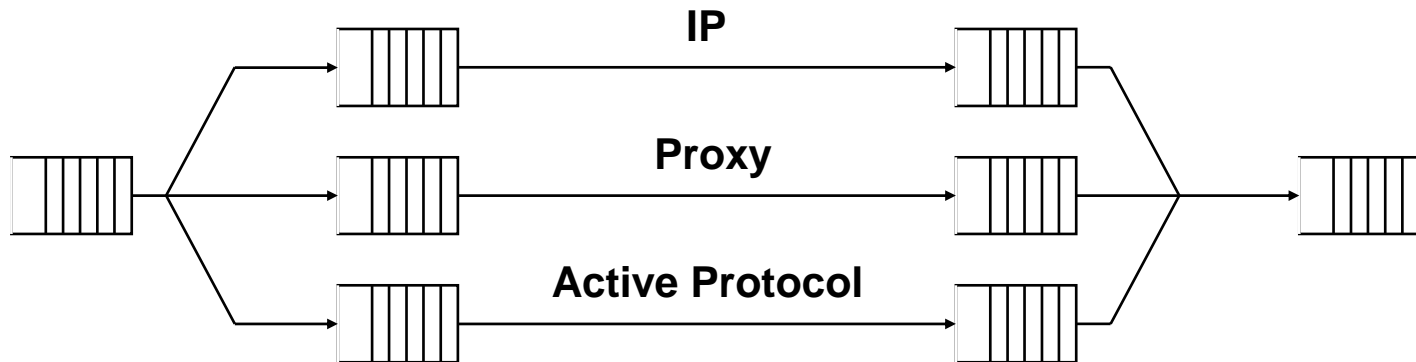


# Virtual Router

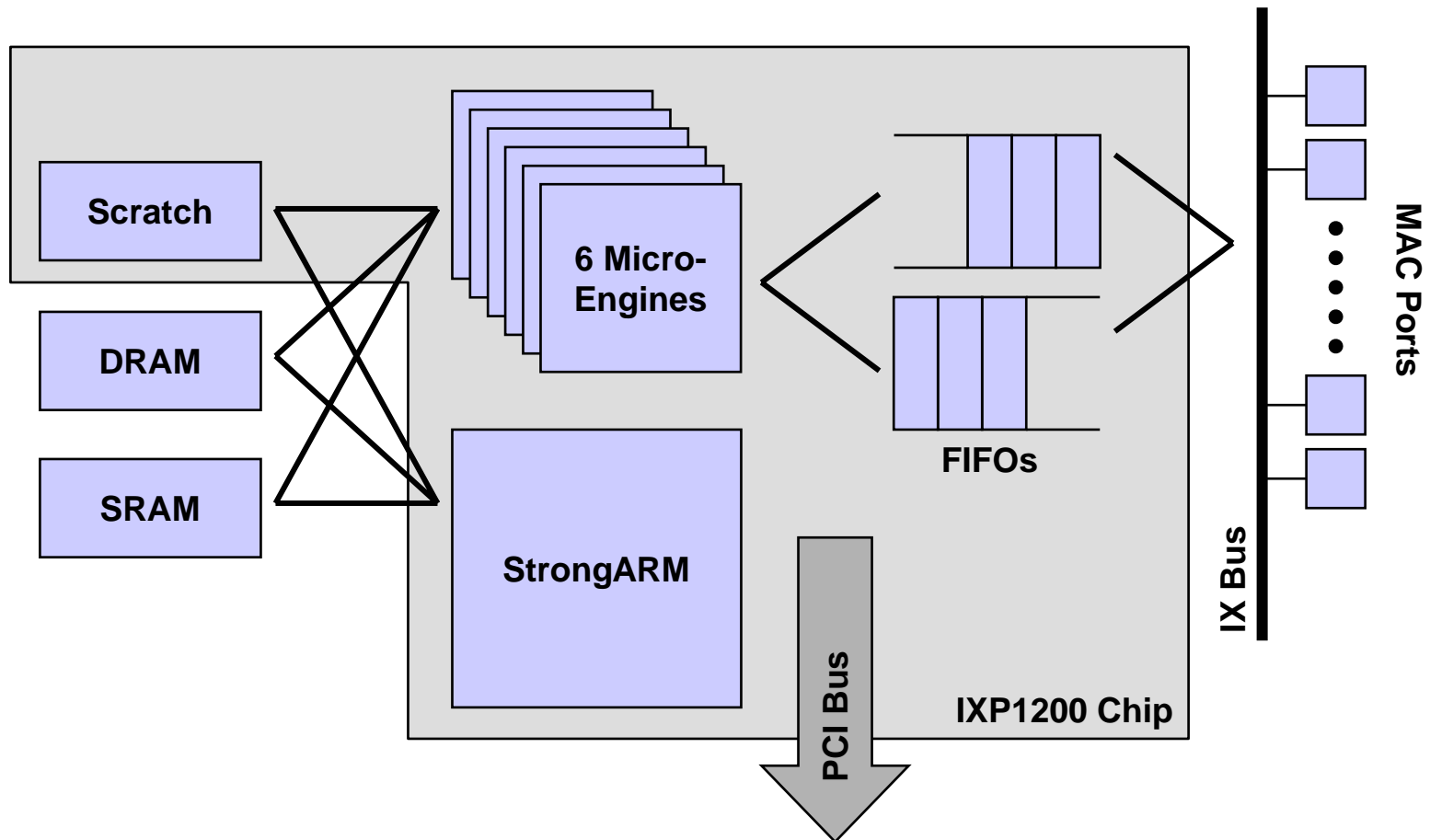
- Classifiers
- Schedulers
- Forwarders



# Simple Example

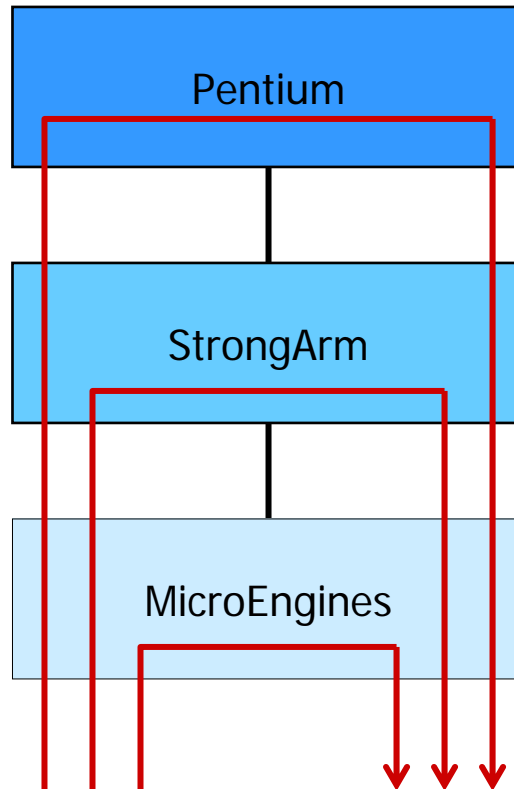


# Intel IXP

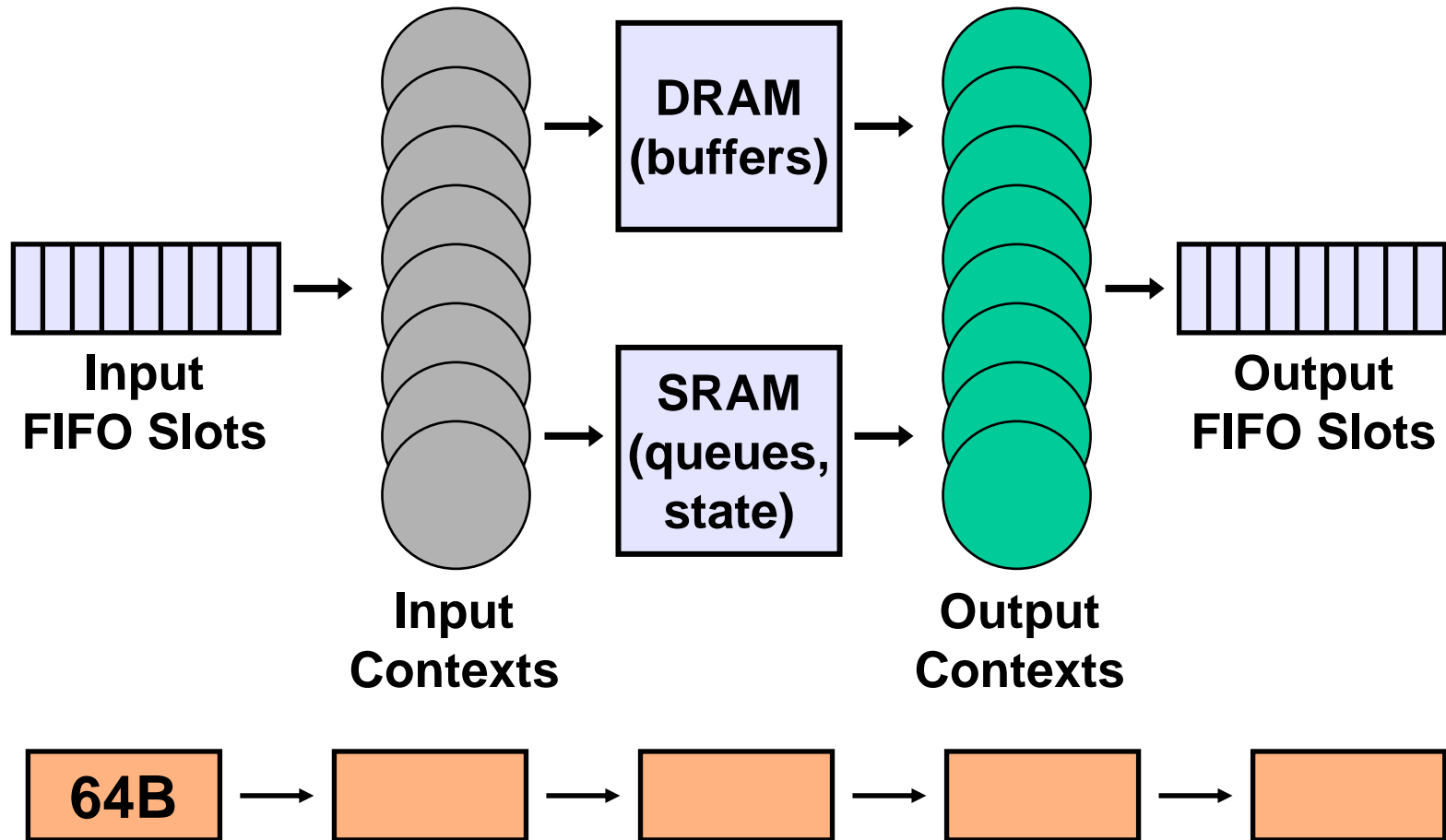




# Processor Hierarchy



# Data Plane Pipeline



# Data Plane Processing

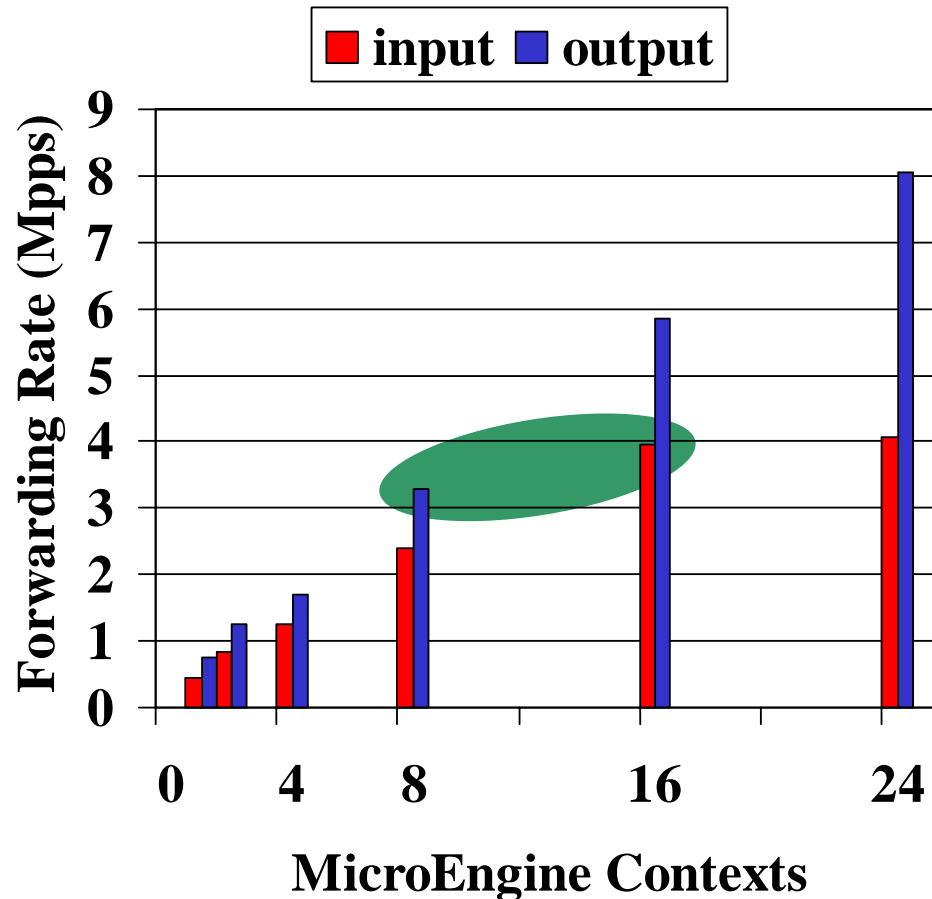
## INPUT context loop

```
wait_for_data
copy in_fifo→regs
Basic_IP_processing
copy regs→DRAM
if (last_fragment)
    enqueue→SRAM
```

## OUTPUT context loop

```
if (need_data)
    select_queue
    dequeue←SRAM
copy DRAM→out_fifo
```

# Pipeline Evaluation



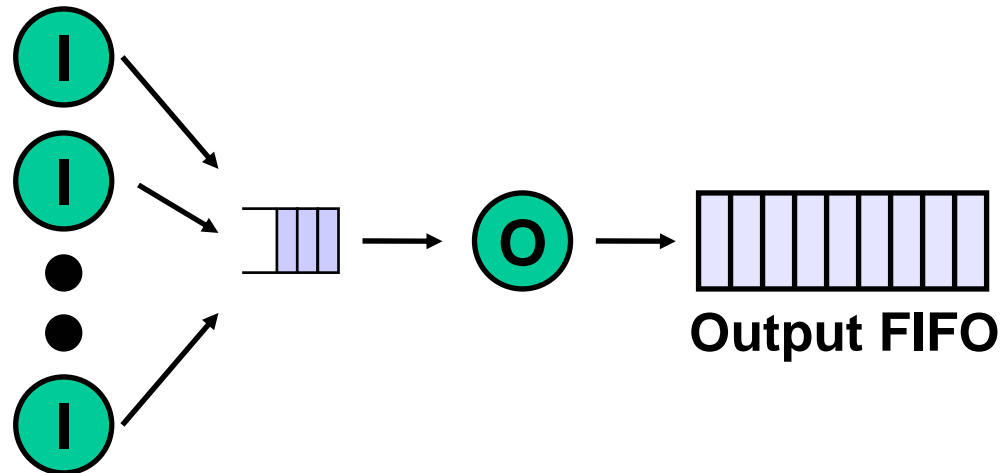
Measured  
independently

100Mbps Ether  $\approx$   
0.142Mpps

# What We Measured

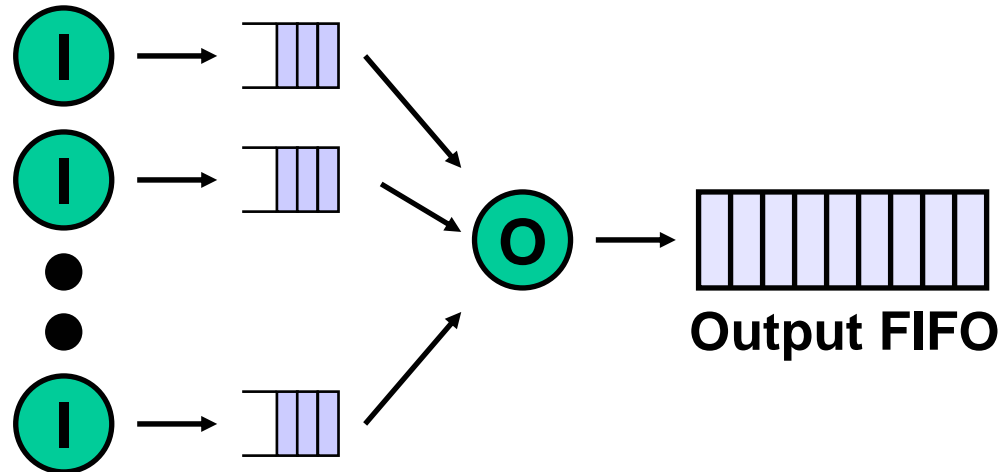
- Static context assignment
  - 16 input / 8 output
- Infinite offered load
- 64-byte (minimum-sized) IP packets
- Three different queuing disciplines

# Single Protected Queue



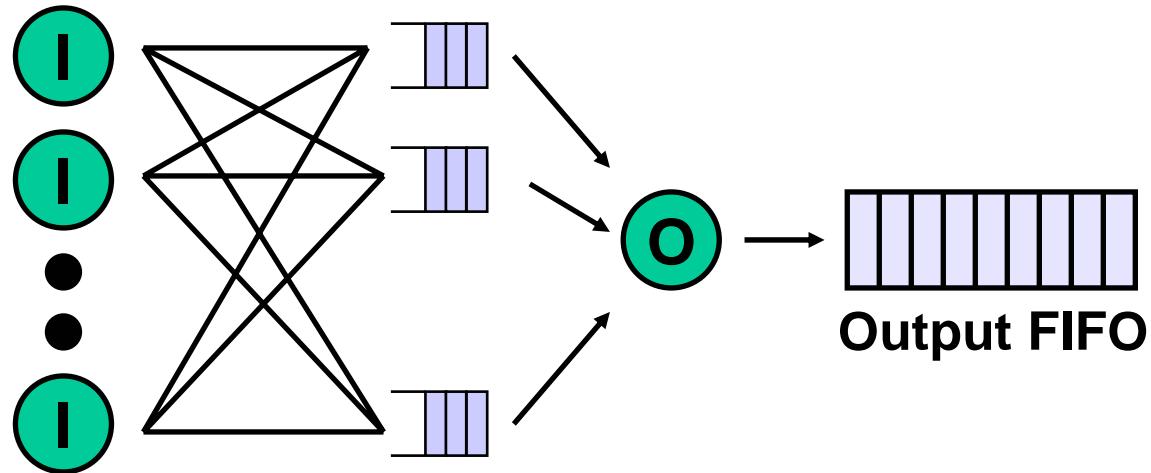
- Lock synchronization
- Max 3.47 Mpps
- Contention lower bound 1.67 Mpps

# Multiple Private Queues



- Output must select queue
- Max 3.29 Mpps

# Multiple Protected Queues



- Output must select queue
- Some QoS scheduling (16 priority levels)
- Max 3.29 Mpps



# Data Plane Processing

## INPUT context loop

```
wait_for_data
copy in_fifo→regs
Basic_IP_processing
copy regs→DRAM
if (last_fragment)
    enqueue→SRAM
```

## OUTPUT context loop

```
if (need_data)
    select_queue
    dequeue←SRAM
copy DRAM→out_fifo
```

# Cycles to Waste

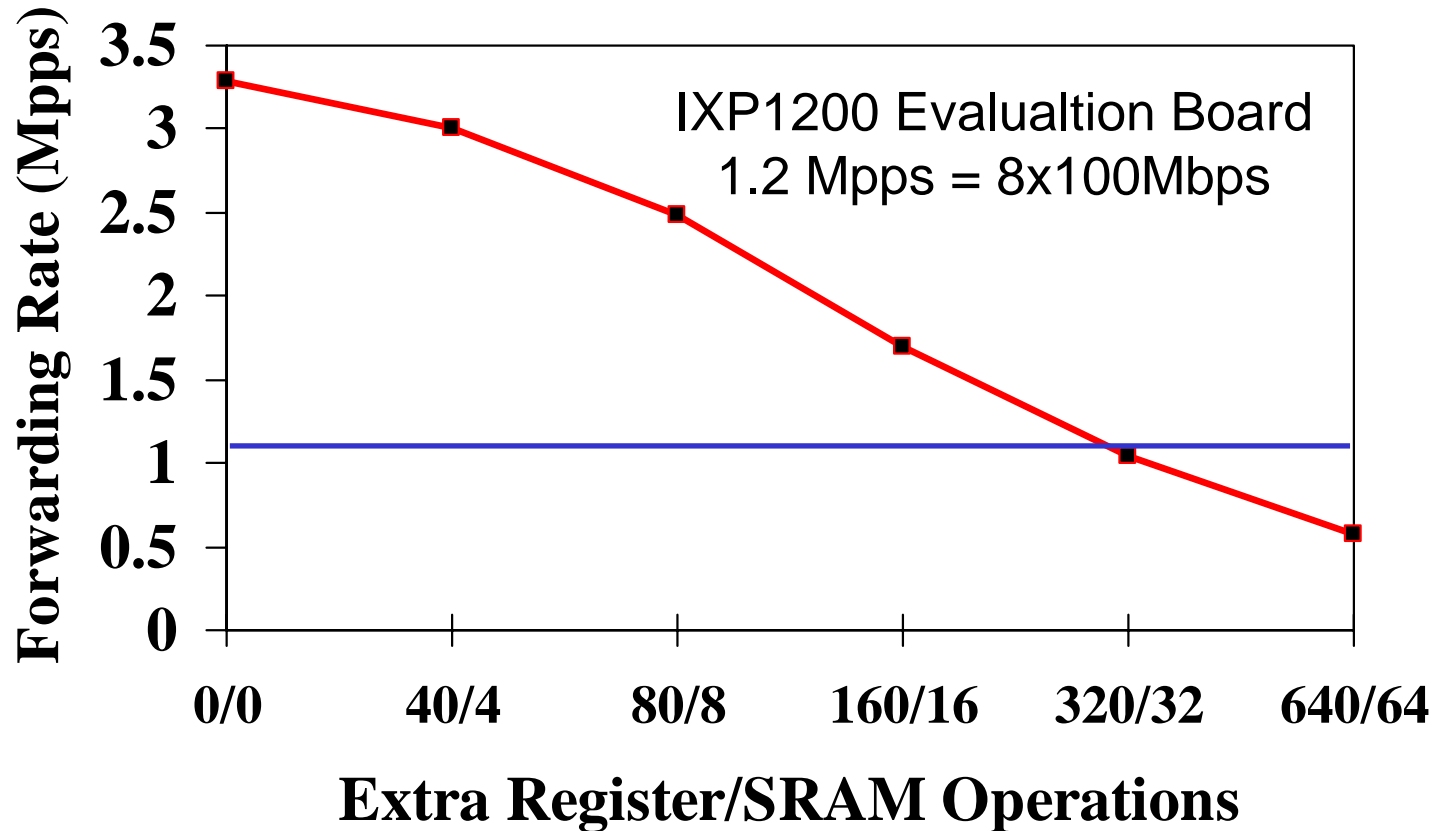
## INPUT context loop

```
wait_for_data
copy in_fifo→regs
Basic_IP_processing
nop
nop
...
nop
copy regs→DRAM
if (last_fragment)
    enqueue→SRAM
```

## OUTPUT context loop

```
if (need_data)
    select_queue
    dequeue←SRAM
copy DRAM→out_fifo
```

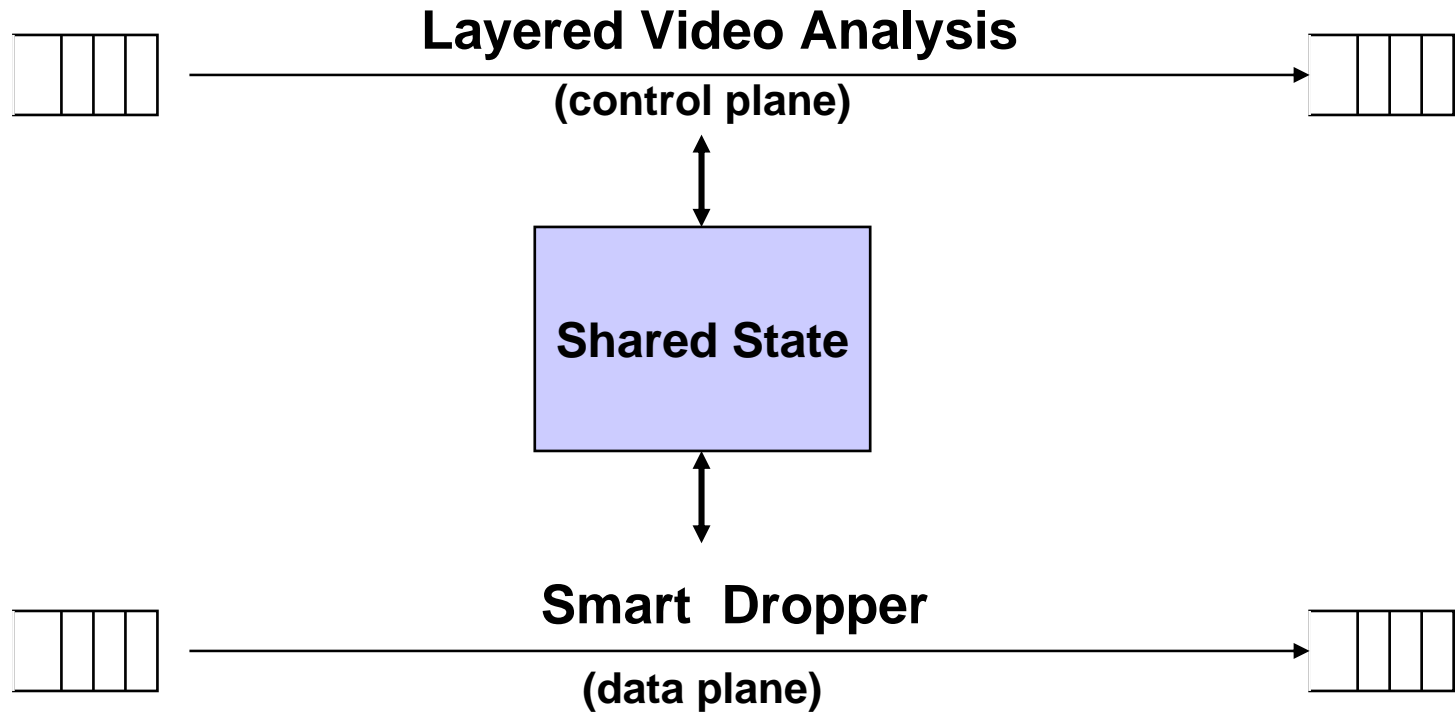
# How Many “NOPs” Possible?



# Data Plane Extensions

Processing	Memory Ops	Register Ops
Basic IP	6	32
TCP Splicer	6	45
TCP SYN Monitor	1	5
ACK Monitor	3	15
Port Filter	5	26
Wavelet Dropper	2	28

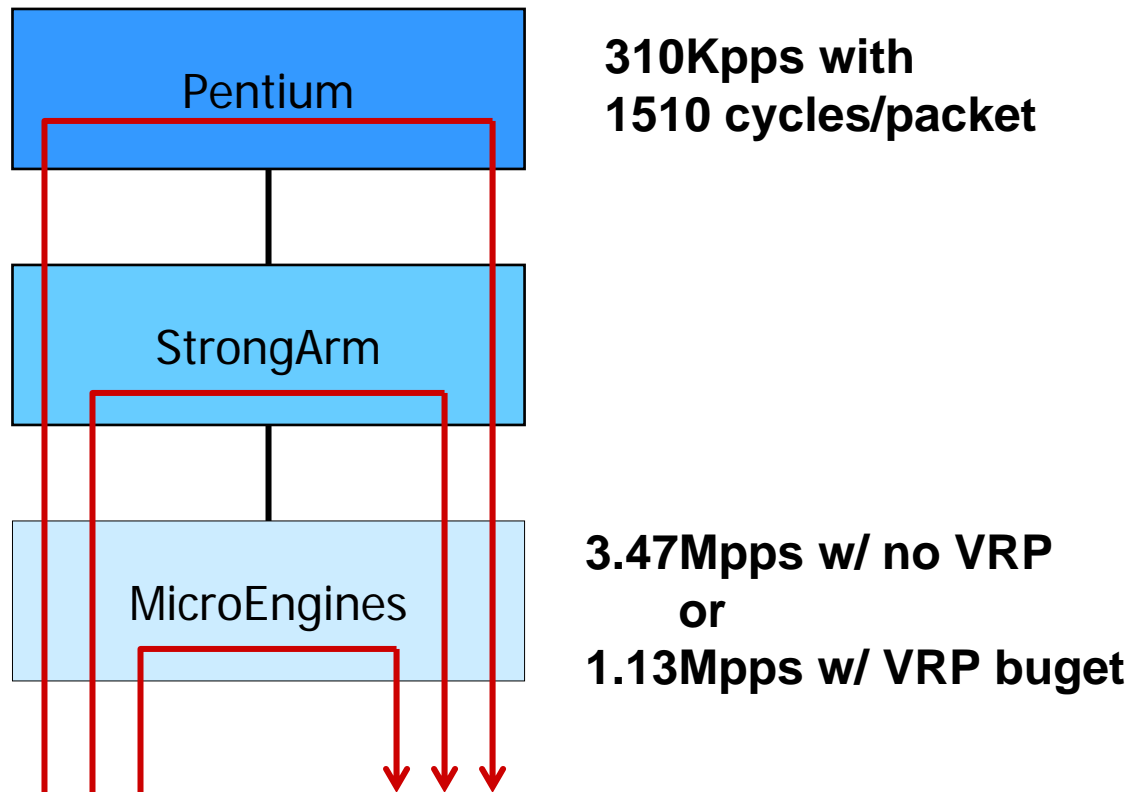
# Control and Data Plane



# What About the StrongARM?

- Shares memory bus with MicroEngines
  - must respect resource budget
- What we do
  - control IXP1200  $\leftrightarrow$  Pentium DMA
  - control MicroEngines
- What might be possible
  - anything within budget
  - exploit instruction and data caches
- We recommend against
  - running Linux

# Performance

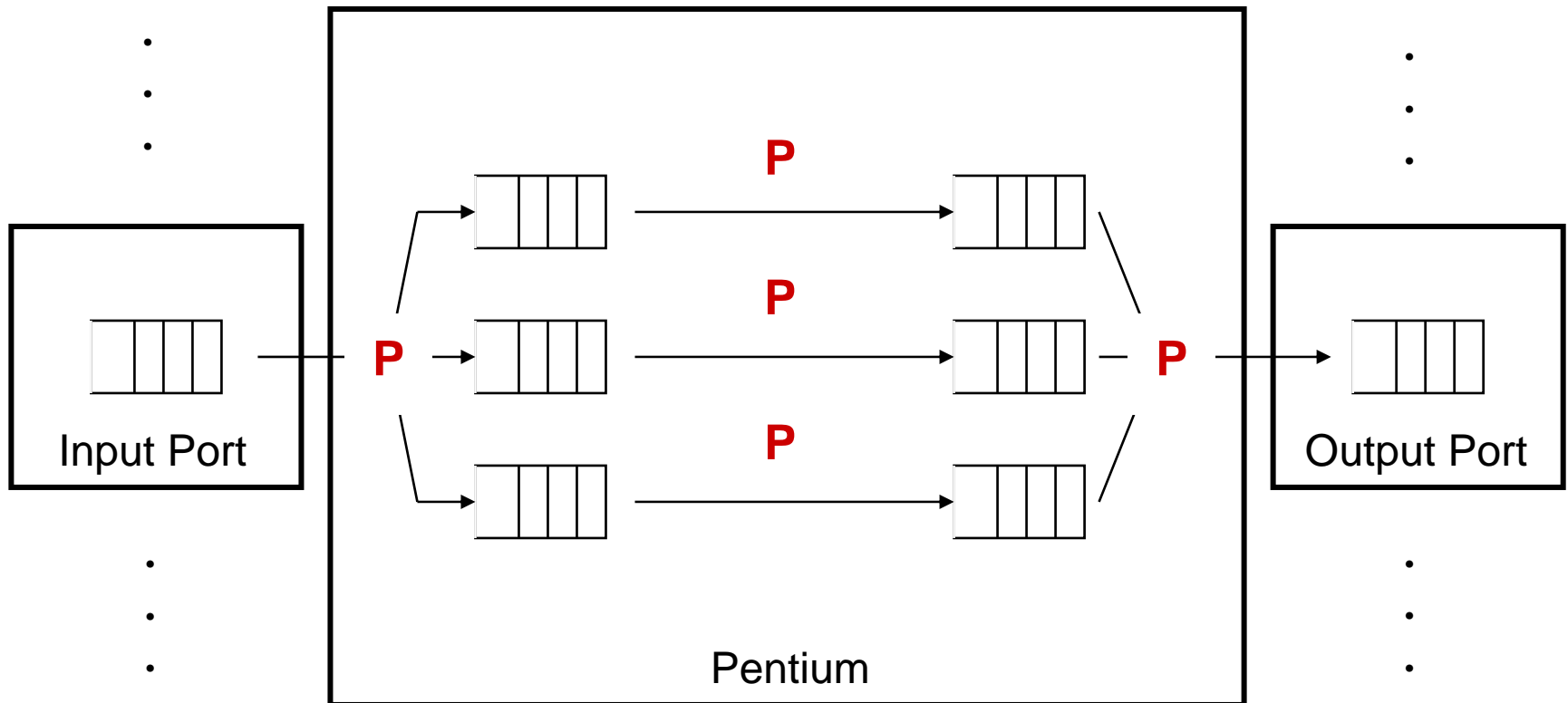


# Pentium

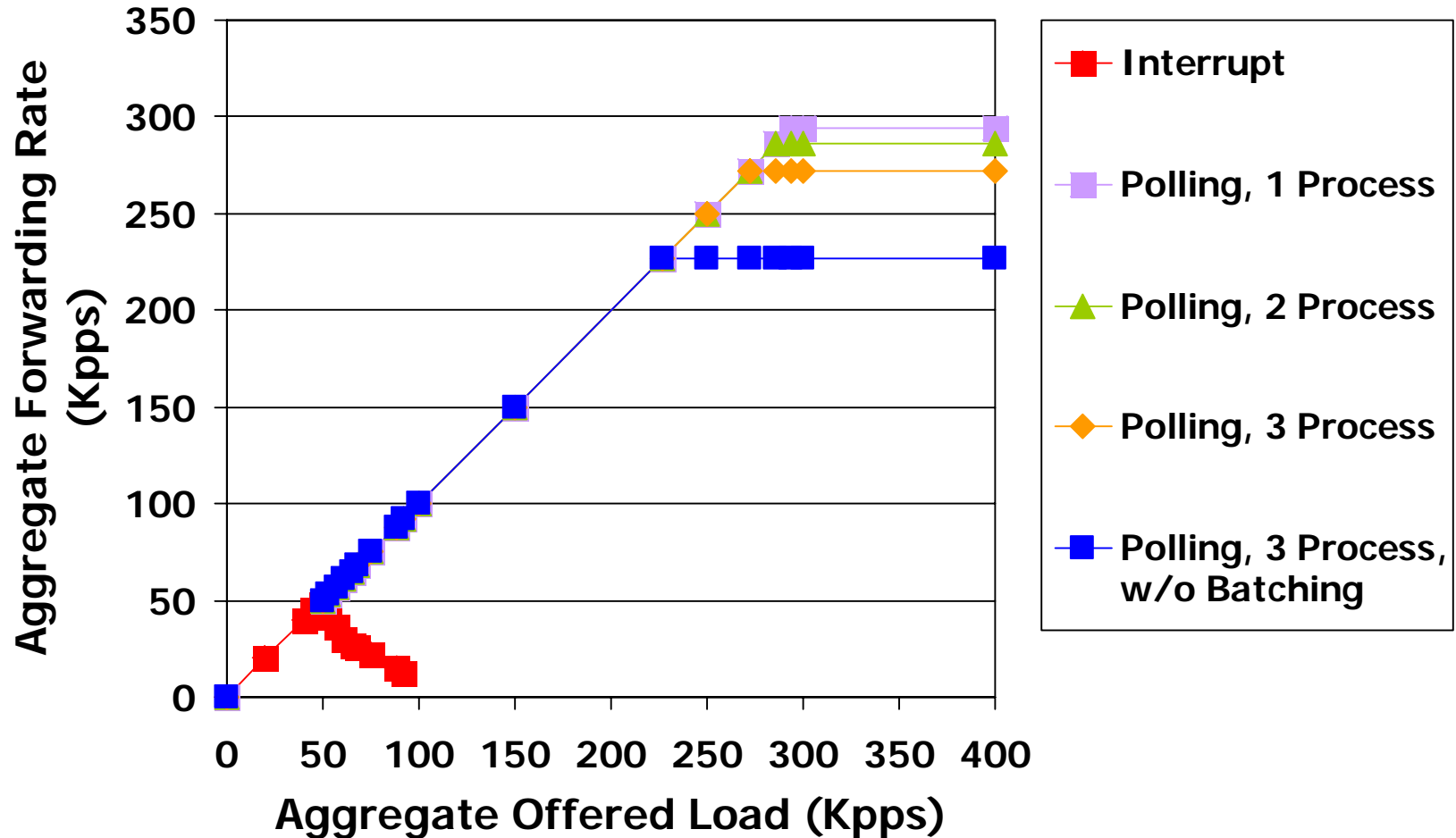
- Runs protocols in the control plane
  - e.g., BGP, OSPF, RSVP
- Run other router extensions
  - e.g., proxies, active protocols, overlays
- Implementation
  - runs Scout OS + Linux IXP driver
  - CPU scheduler is key



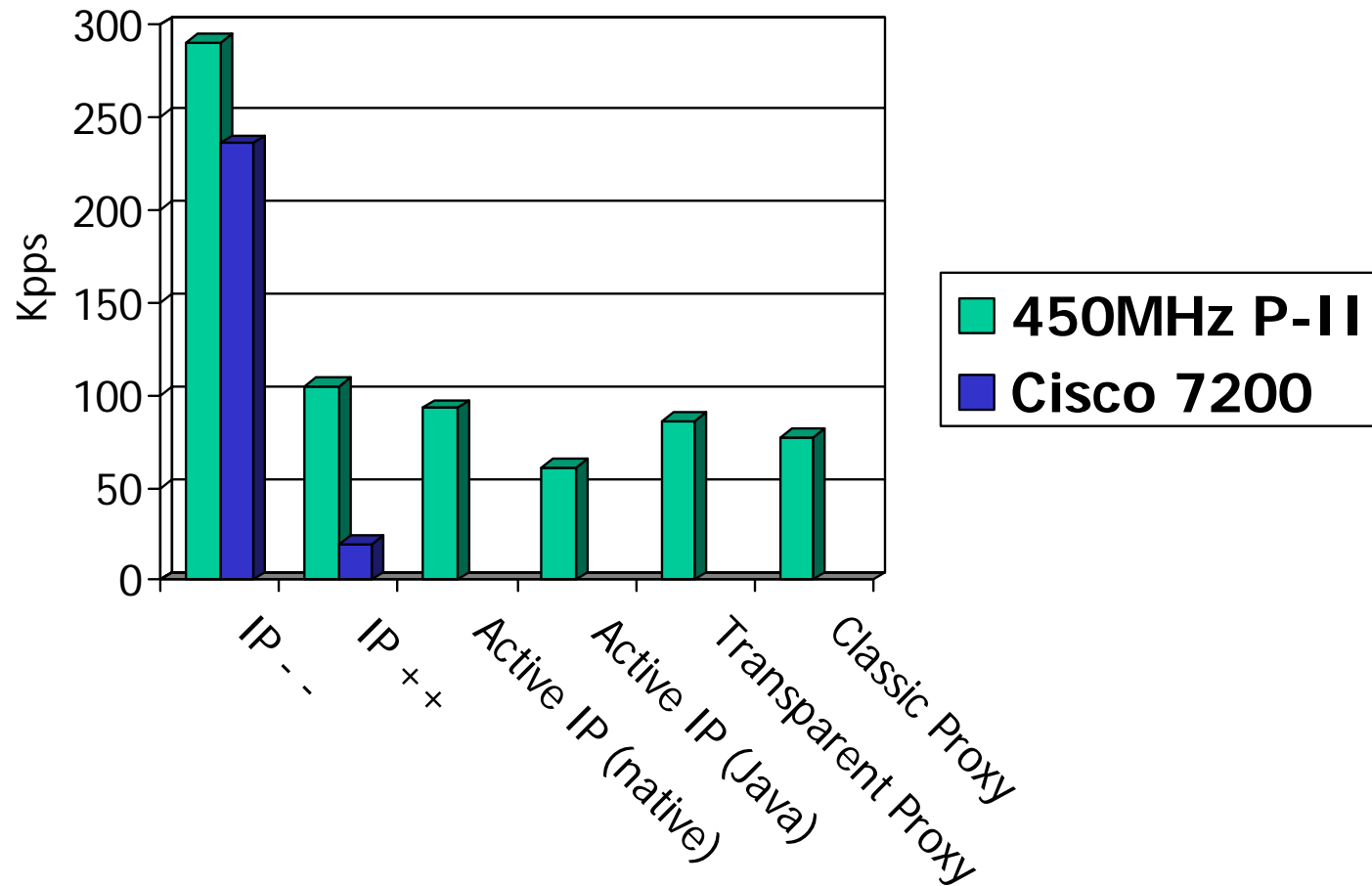
# Processes



# Performance



# Performance (cont)



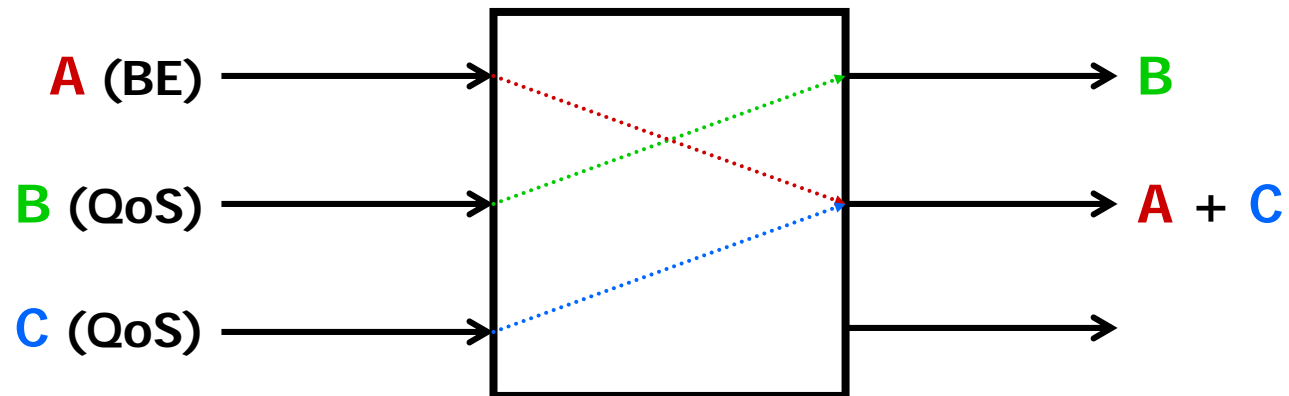
# Scheduling Mechanism

- Proportional share forms the base
  - each process reserves a cycle rate
  - provides isolation between processes
  - unused capacity fairly distributed
- Eligibility
  - a process receives its share only when its source queue is not empty and sink queue is not full
- Batching
  - to minimize context switch overhead

# Share Assignment

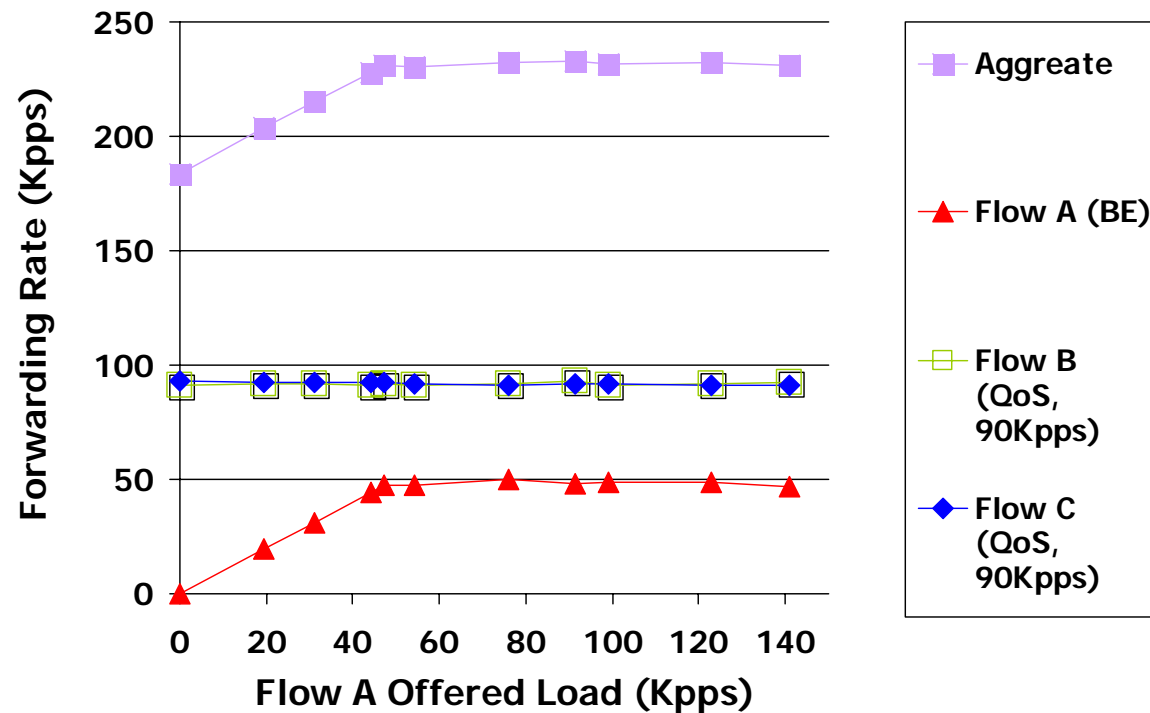
- QoS Flows
  - assume link rate is given, derive cycle rate
  - conservative rate to input process
  - keep batching level low
- Best Effort Flows
  - may be influenced by admin policy
  - use shares to balance system (avoid livelock)
  - keep batching level high

# Experiment



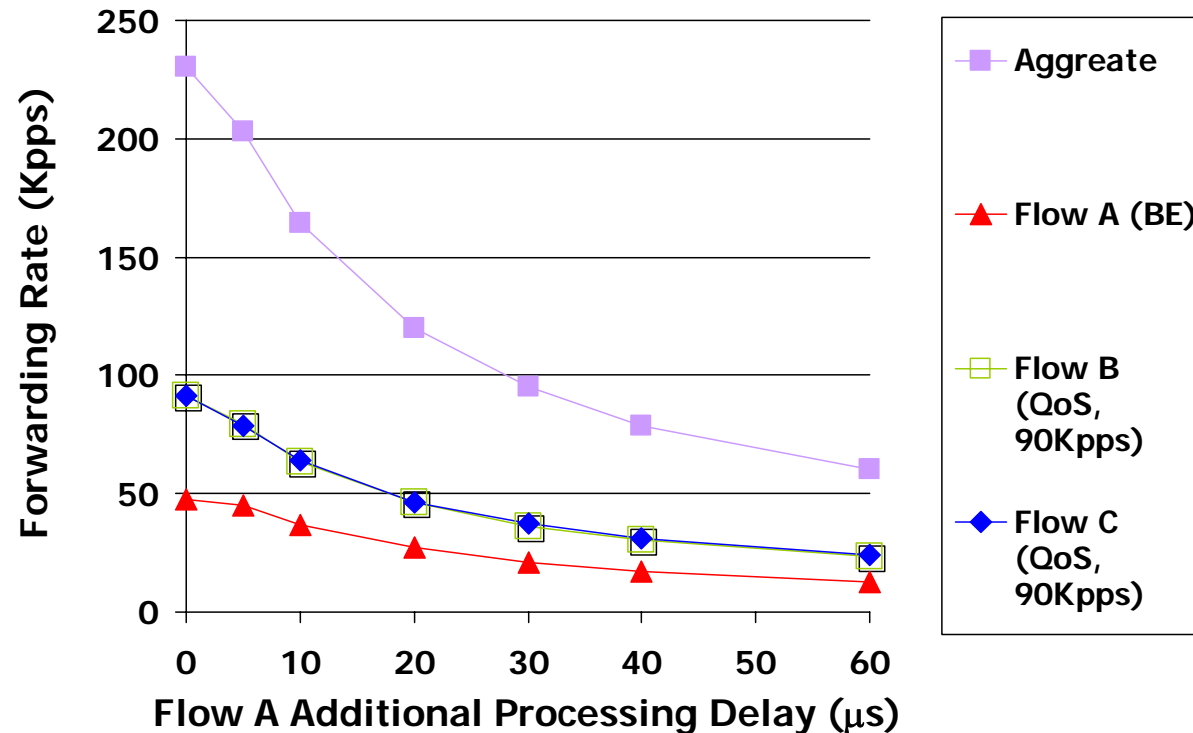
# Mixing Best Effort and QoS

- Increase offered load from **A**



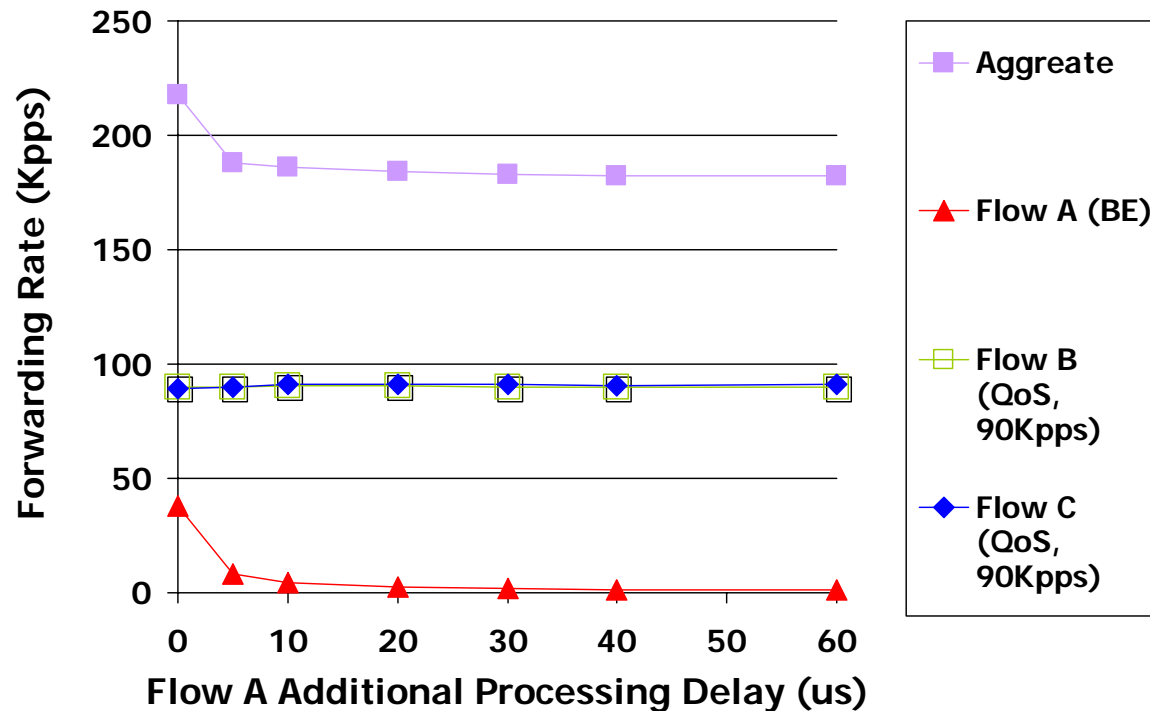
# CPU vs Link

- Fix **A** at 50Kpps, increase its processing cost



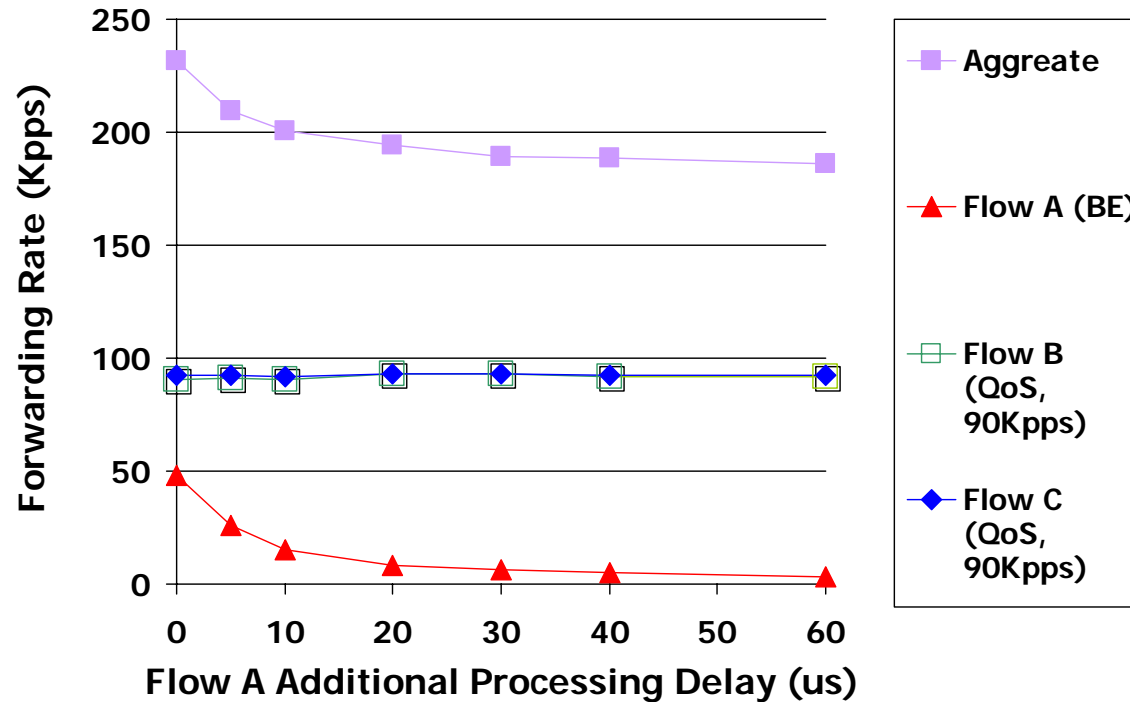


# Turn Batching Off



- CPU efficiency: 66.2%

# Enforce Time Slice



- CPU efficiency: 81.6% (30us quantum)

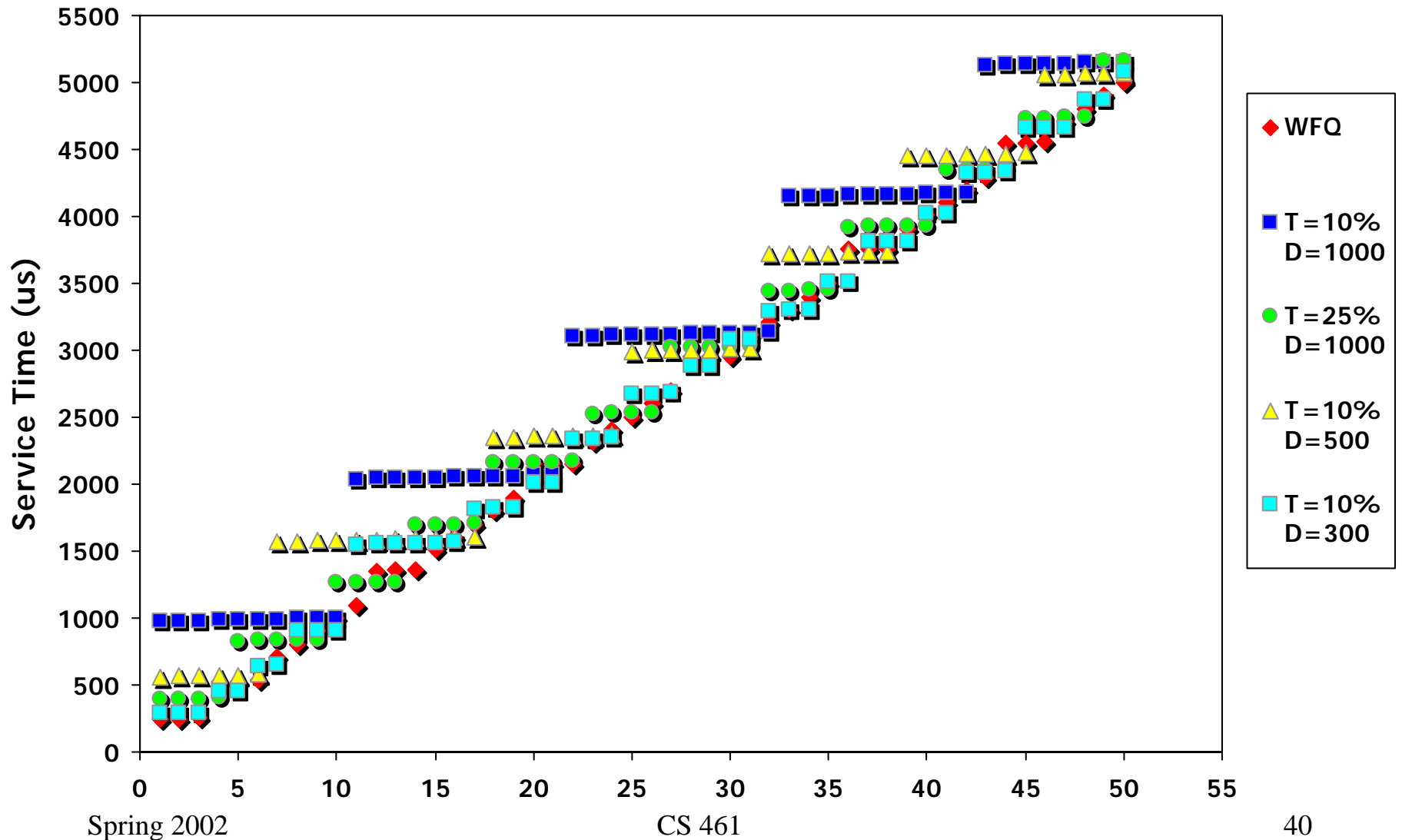
# Batching Throttle

- Scheduler Granularity:  $G$ 
  - flow processes as many packets as possible w/in  $G$
- Efficiency Index:  $E$ , Overhead Threshold:  $T$ 
  - keep the overhead under  $T\%$ , then  $1 / (1+T) < E$
- Batch Threshold:  $B_i$ 
  - don't consider Flow  $i$  active until it has accumulated at least  $B_i$  packets, where  $C_{sw} / (B_i \times C_i) < T$
- Delay Threshold:  $D_i$ 
  - consider a flow that has waited  $D_i$  active

# Dynamic Control

- Flow specifies delay requirement  $D$
- Measure context switch overhead offline
- Record average flow runtime
- Set  $E$  based on workload
- Calculate batch-level  $B$  for flow

# Packet Trace



# *TCP and UDP*

## *(End-to-End Protocols)*

*Go To Talk Outline*

# Reliable Byte-Stream (TCP)

## Outline

- Connection Establishment/Termination
- Sliding Window Revisited
- Flow Control
- Adaptive Timeout

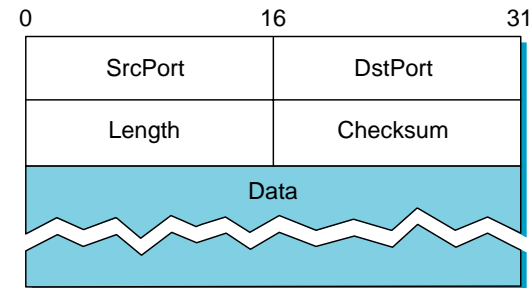
# End-to-End Protocols

- Underlying best-effort network
  - drop messages
  - re-orders messages
  - delivers duplicate copies of a given message
  - limits messages to some finite size
  - delivers messages after an arbitrarily long delay
- Common end-to-end services
  - guarantee message delivery
  - deliver messages in the same order they are sent
  - deliver at most one copy of each message
  - support arbitrarily large messages
  - support synchronization
  - allow the receiver to flow control the sender
  - support multiple application processes on each host



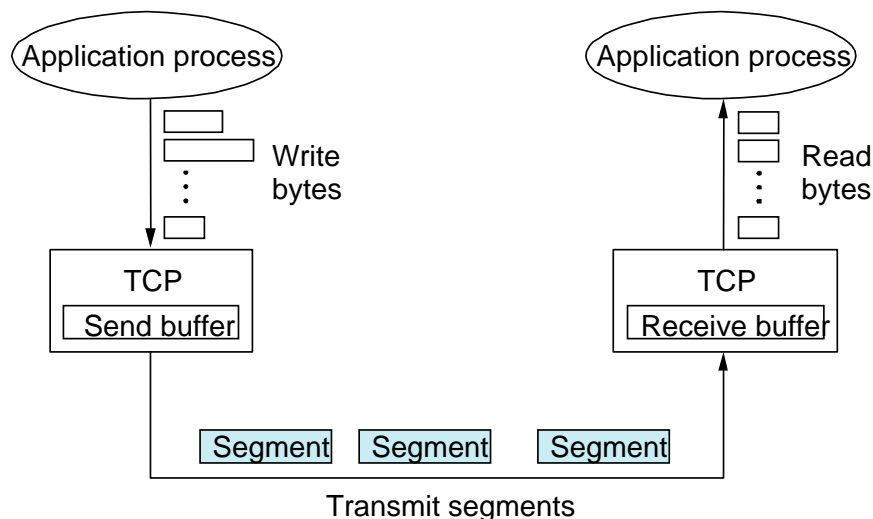
# Simple Demultiplexor (UDP)

- Unreliable and unordered datagram service
- Adds multiplexing
- No flow control
- Endpoints identified by ports
  - servers have *well-known* ports
  - see **/etc/services** on Unix
- Header format
- Optional checksum
  - psuedo header + UDP header + data



# TCP Overview

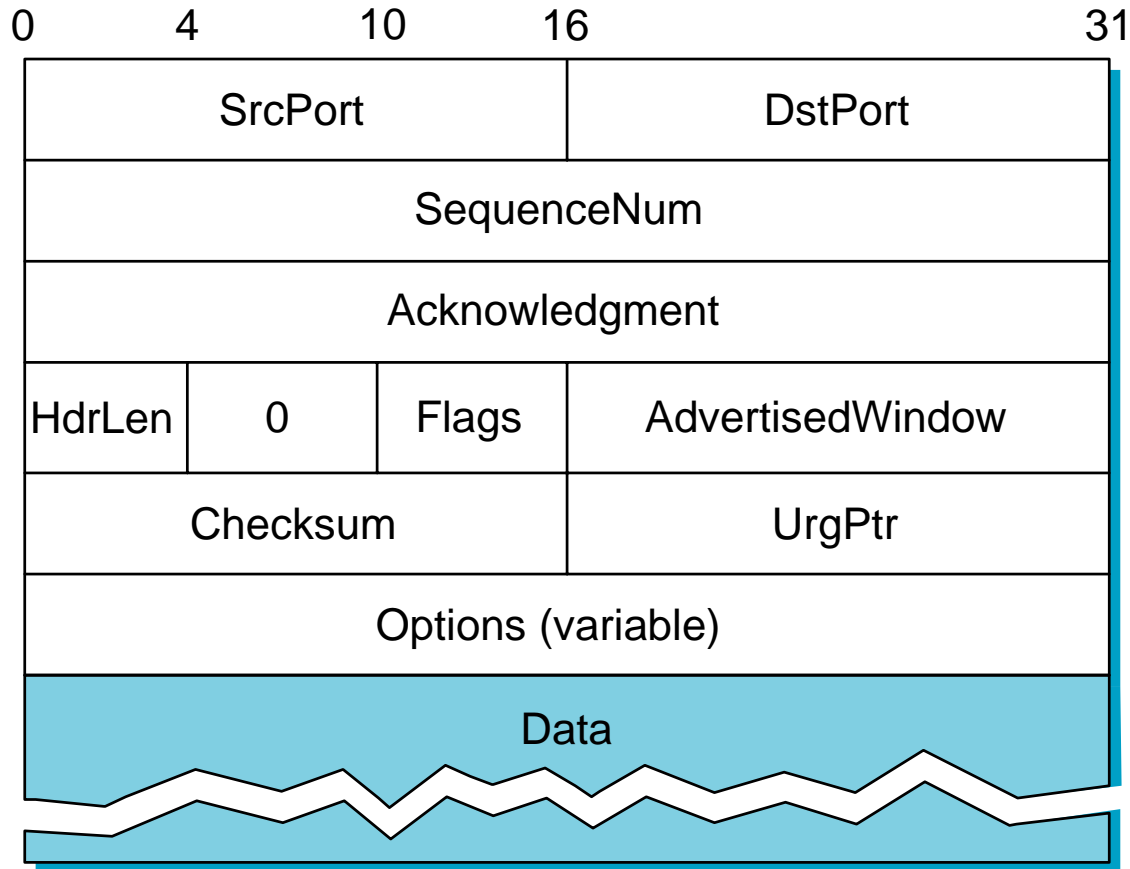
- Connection-oriented
- Byte-stream
  - app writes bytes
  - TCP sends *segments*
  - app reads bytes
- Full duplex
- Flow control: keep sender from overrunning receiver
- Congestion control: keep sender from overrunning network



# Data Link Versus Transport

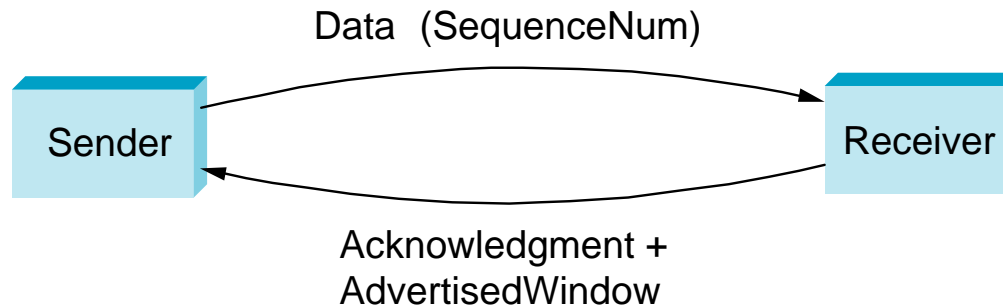
- Potentially connects many different hosts
  - need explicit connection establishment and termination
- Potentially different RTT
  - need adaptive timeout mechanism
- Potentially long delay in network
  - need to be prepared for arrival of very old packets
- Potentially different capacity at destination
  - need to accommodate different node capacity
- Potentially different network capacity
  - need to be prepared for network congestion

# Segment Format



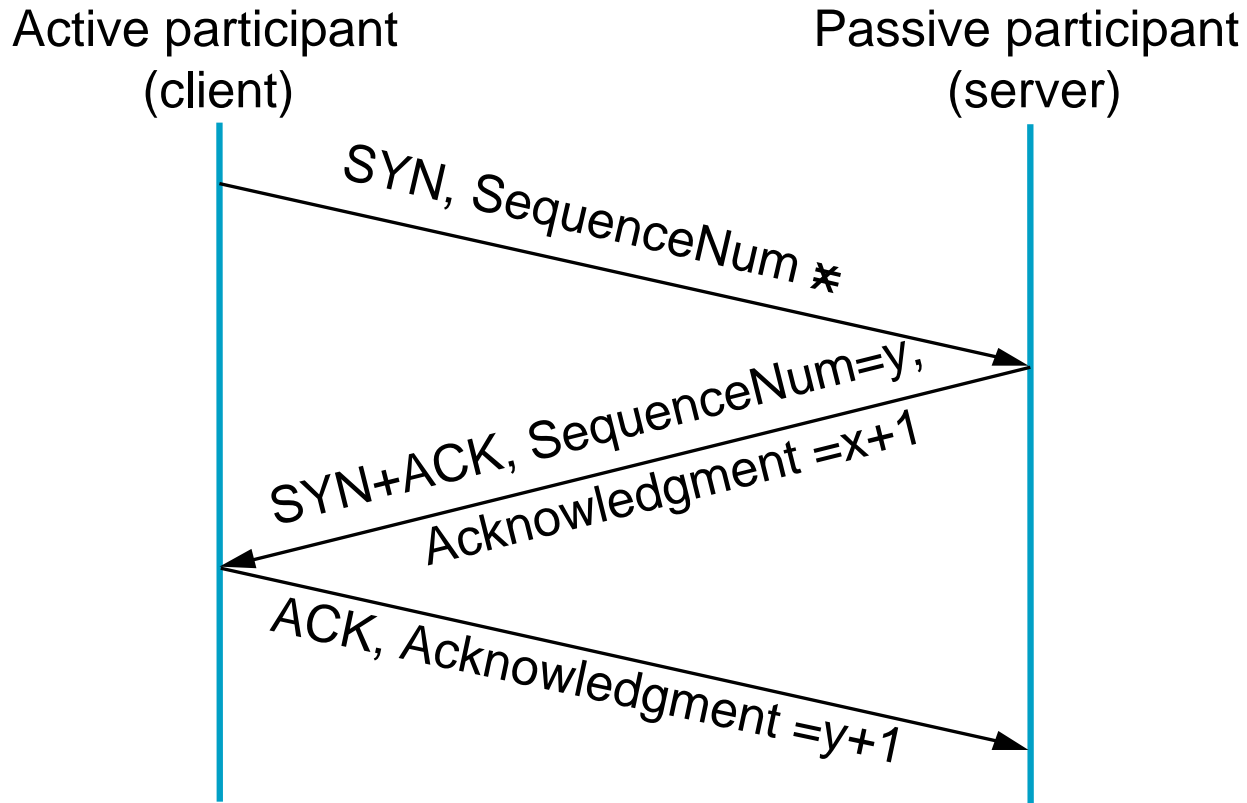
# Segment Format (cont)

- Each connection identified with 4-tuple:
  - **(SrcPort, SrcIPAddr, DsrPort, DstIPAddr)**
- Sliding window + flow control
  - **acknowledgment, SequenceNum, AdvertisedWinow**

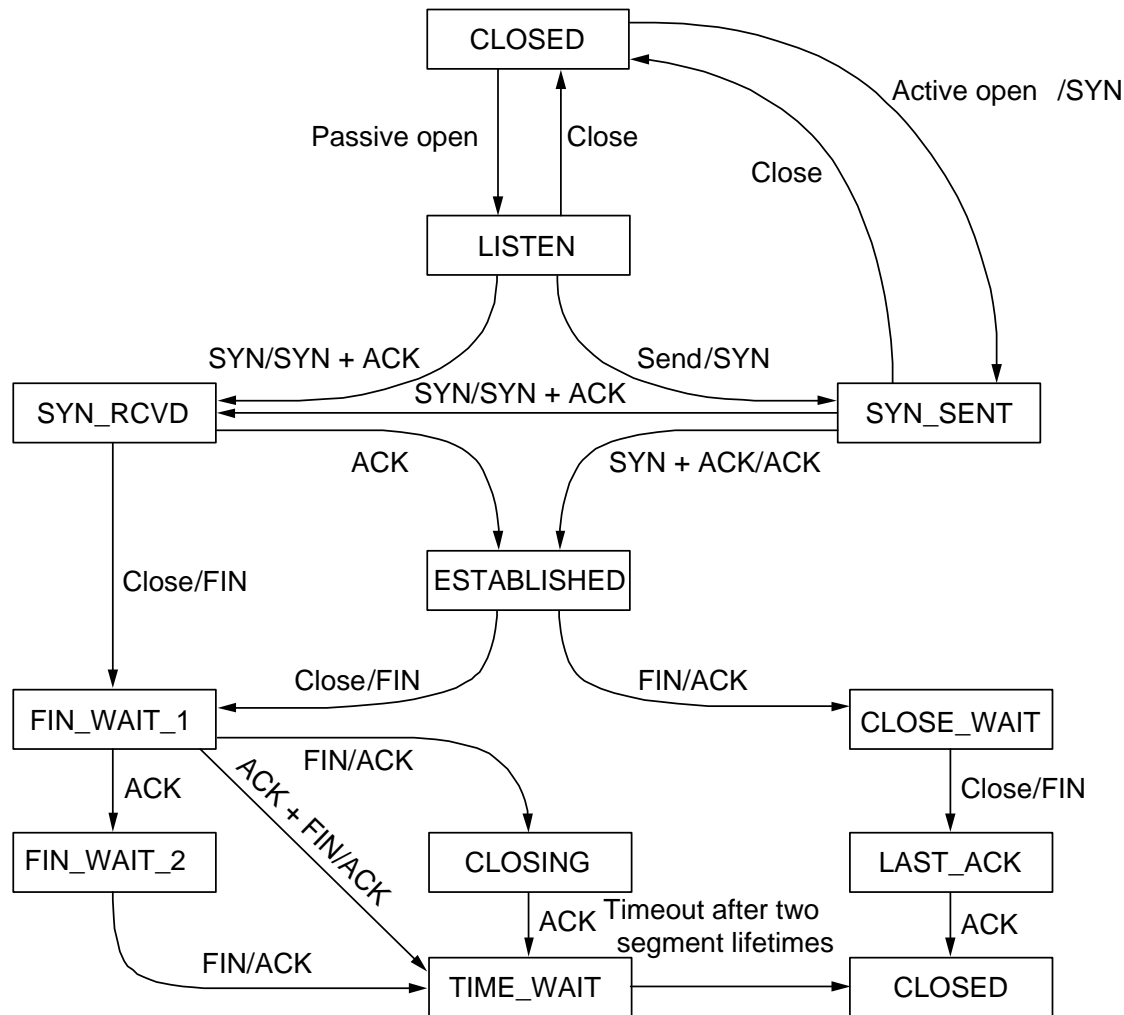


- **Flags**
  - **SYN, FIN, RESET, PUSH, URG, ACK**
- **Checksum**
  - pseudo header + TCP header + data

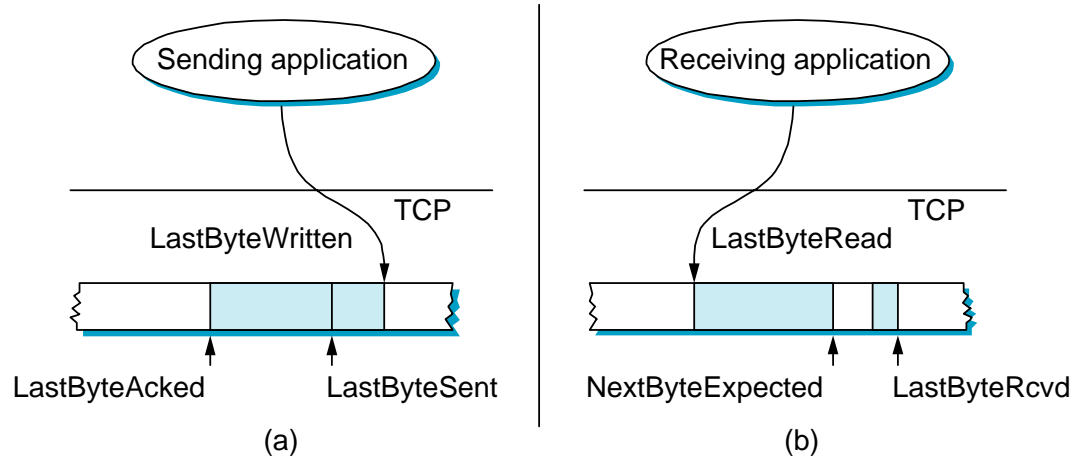
# Connection Establishment and Termination



# State Transition Diagram



# Sliding Window Revisited



- Sending side

- $\text{LastByteAcked} \leq \text{LastByteSent}$
- $\text{LastByteSent} \leq \text{LastByteWritten}$
- buffer bytes between  $\text{LastByteAcked}$  and  $\text{LastByteWritten}$

- Receiving side

- $\text{LastByteRead} < \text{NextByteExpected}$
- $\text{NextByteExpected} \leq \text{LastByteRcvd} + 1$
- buffer bytes between  $\text{NextByteRead}$  and  $\text{LastByteRcvd}$

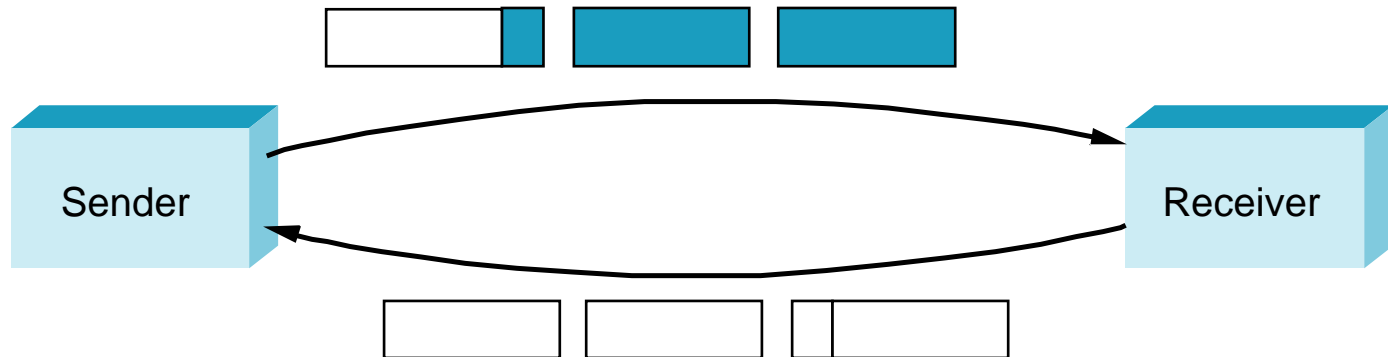


# Flow Control

- Send buffer size: **MaxSendBuffer**
- Receive buffer size: **MaxRcvBuffer**
- Receiving side
  - **LastByteRcvd - LastByteRead  $\leq$  MaxRcvBuffer**
  - **AdvertisedWindow = MaxRcvBuffer - (NextByteExpected - NextByteRead)**
- Sending side
  - **LastByteSent - LastByteAcked  $\leq$  AdvertisedWindow**
  - **EffectiveWindow = AdvertisedWindow - (LastByteSent - LastByteAcked)**
  - **LastByteWritten - LastByteAcked  $\leq$  MaxSendBuffer**
  - block sender if **(LastByteWritten - LastByteAcked) + y > MaxSenderBuffer**
- Always send ACK in response to arriving data segment
- Persist when **AdvertisedWindow = 0**

# Silly Window Syndrome

- How aggressively does sender exploit open window?



- Receiver-side solutions
  - after advertising zero window, wait for space equal to a maximum segment size (MSS)
  - delayed acknowledgements

# Nagle's Algorithm

- How long does sender delay sending data?
  - too long: hurts interactive applications
  - too short: poor network utilization
  - strategies: timer-based vs self-clocking
- When application generates additional data
  - if fills a max segment (and window open): send it
  - else
    - if there is unack'ed data in transit: buffer it until ACK arrives
    - else: send it

# Protection Against Wrap Around

- 32-bit **SequenceNum**

Bandwidth	Time Until Wrap Around
T1 (1.5 Mbps)	6.4 hours
Ethernet (10 Mbps)	57 minutes
T3 (45 Mbps)	13 minutes
FDDI (100 Mbps)	6 minutes
STS-3 (155 Mbps)	4 minutes
STS-12 (622 Mbps)	55 seconds
STS-24 (1.2 Gbps)	28 seconds

# Keeping the Pipe Full

- 16-bit **AdvertisedWindow**

Bandwidth	Delay x Bandwidth Product
T1 (1.5 Mbps)	18KB
Ethernet (10 Mbps)	122KB
T3 (45 Mbps)	549KB
FDDI (100 Mbps)	1.2MB
STS-3 (155 Mbps)	1.8MB
STS-12 (622 Mbps)	7.4MB
STS-24 (1.2 Gbps)	14.8MB

assuming 100ms RTT

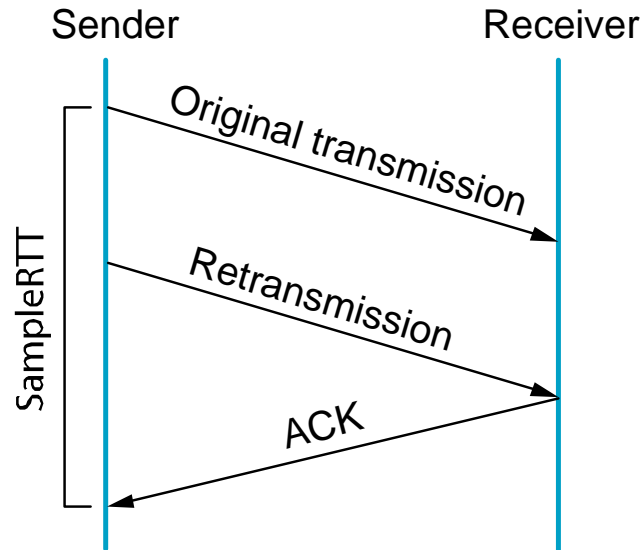
# TCP Extensions

- Implemented as header options
- Store timestamp in outgoing segments
- Extend sequence space with 32-bit timestamp (PAWS)
- Shift (scale) advertised window

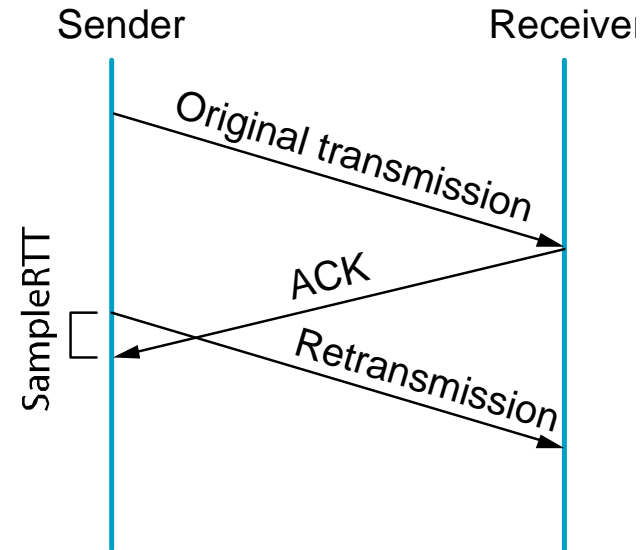
# Adaptive Retransmission (Original Algorithm)

- Measure **SampleRTT** for each segment / ACK pair
- Compute weighted average of RTT
  - **EstRTT** =  $\alpha \times \text{EstRTT} + \beta \times \text{SampleRTT}$
  - where  $\alpha + \beta = 1$
  - $\alpha$  between 0.8 and 0.9
  - $\beta$  between 0.1 and 0.2
- Set timeout based on **EstRTT**
  - **TimeOut** =  $2 \times \text{EstRTT}$

# Karn/Partridge Algorithm



(a)



(b)

- Do not sample RTT when retransmitting
- Double timeout after each retransmission



# Jacobson/ Karels Algorithm

- New Calculations for average RTT
- **Diff** = **SampleRTT** - **EstRTT**
- **EstRTT** = **EstRTT** + ( $\delta \times \text{Diff}$ )
- **Dev** = **Dev** +  $\delta (|\text{Diff}| - \text{Dev})$ 
  - where  $\delta$  is a factor between 0 and 1
- Consider variance when setting timeout value
- **TimeOut** =  $\mu \times \text{EstRTT} + \phi \times \text{Dev}$ 
  - where  $\mu = 1$  and  $\phi = 4$
- Notes
  - algorithm only as good as granularity of clock (500ms on Unix)
  - accurate timeout mechanism important to congestion control (later)

# *Congestion Control*

*Go To Talk Outline*

# Congestion Control

## Outline

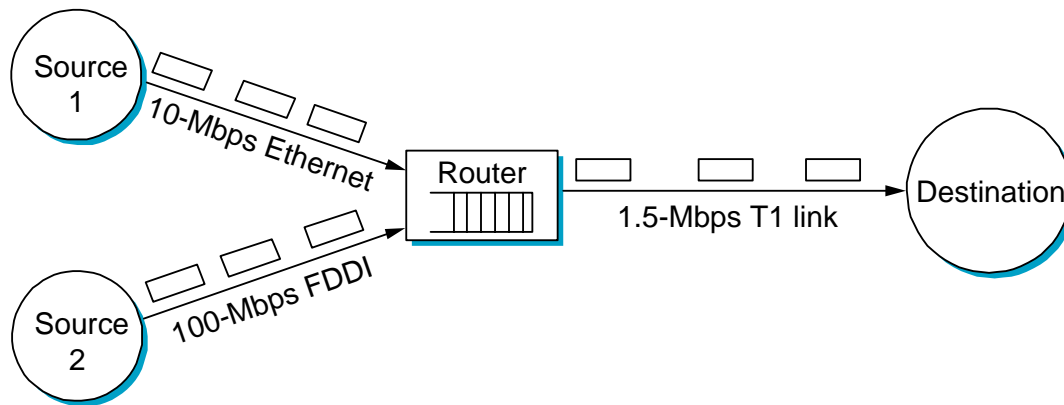
Queuing Discipline

Reacting to Congestion

Avoiding Congestion

# Issues

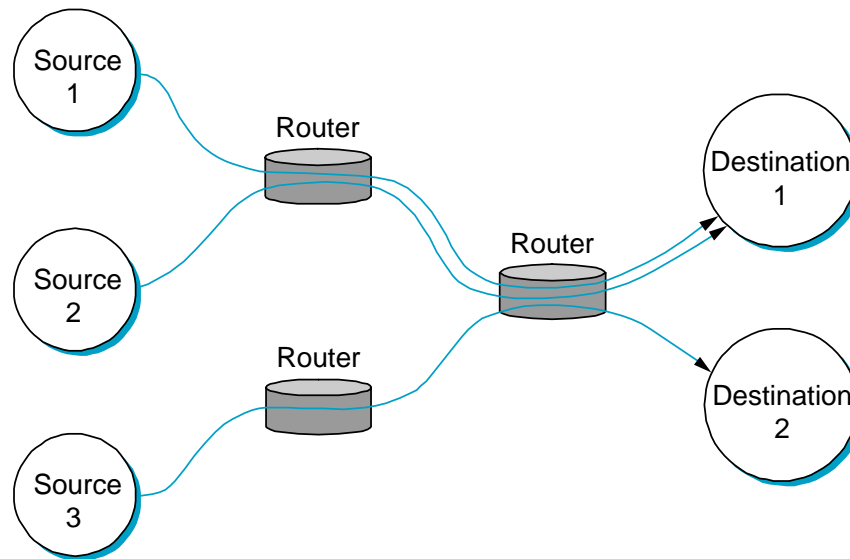
- Two sides of the same coin
  - pre-allocate resources so as to avoid congestion
  - control congestion if (and when) it occurs



- Two points of implementation
  - hosts at the edges of the network (transport protocol)
  - routers inside the network (queuing discipline)
- Underlying service model
  - best-effort (assume for now)
  - multiple *qualities of service* (later)

# Framework

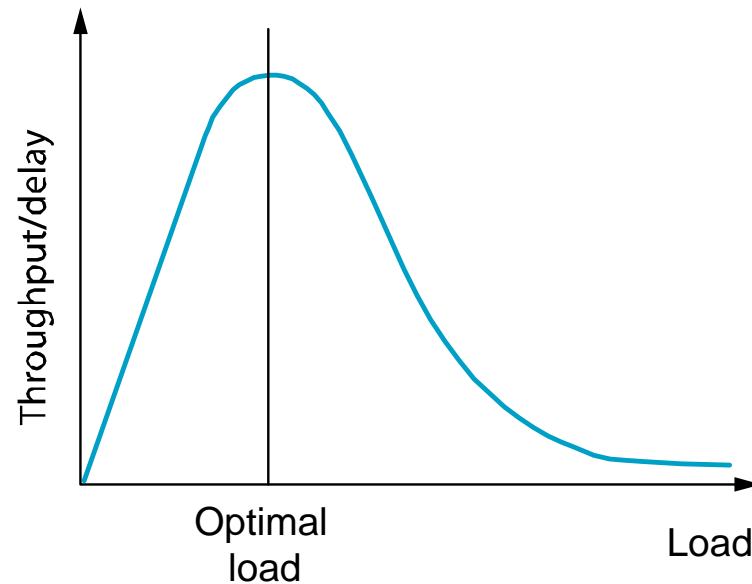
- Connectionless flows
  - sequence of packets sent between source/destination pair
  - maintain *soft state* at the routers



- Taxonomy
  - router-centric versus host-centric
  - reservation-based versus feedback-based
  - window-based versus rate-based

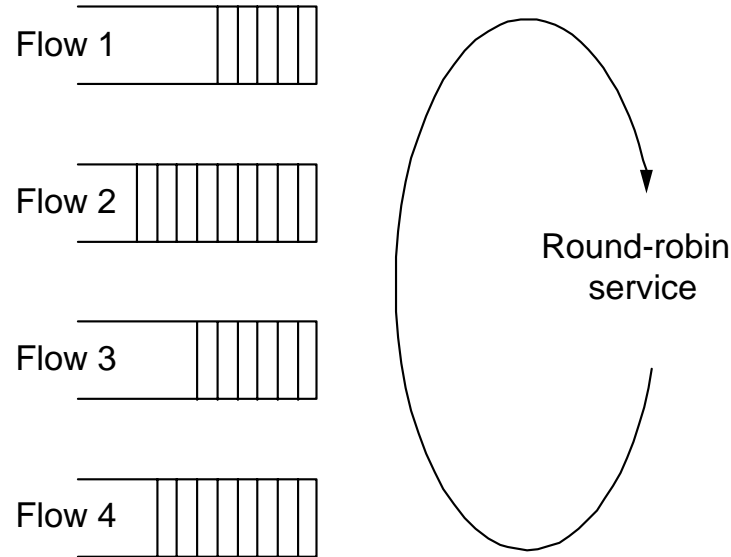
# Evaluation

- Fairness
- Power (ratio of throughput to delay)



# Queuing Discipline

- First-In-First-Out (FIFO)
  - does not discriminate between traffic sources
- Fair Queuing (FQ)
  - explicitly segregates traffic based on flows
  - ensures no flow captures more than its share of capacity
  - variation: weighted fair queuing (WFQ)
- Problem?



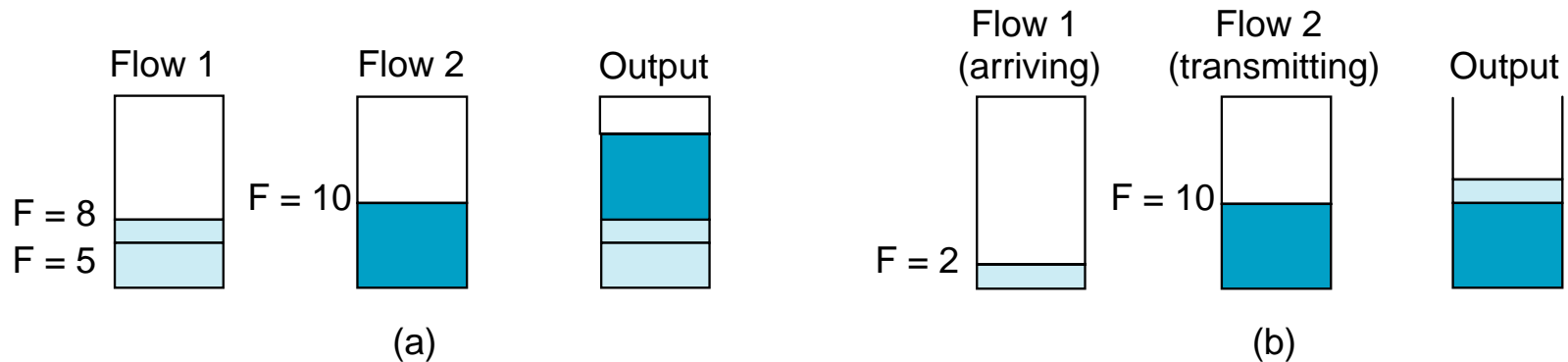
# FQ Algorithm

- Suppose clock ticks each time a bit is transmitted
- Let  $P_i$  denote the length of packet  $i$
- Let  $S_i$  denote the time when start to transmit packet  $i$
- Let  $F_i$  denote the time when finish transmitting packet  $i$
- $F_i = S_i + P_i$
- When does router start transmitting packet  $i$ ?
  - if before router finished packet  $i - 1$  from this flow, then immediately after last bit of  $i - 1$  ( $F_{i-1}$ )
  - if no current packets for this flow, then start transmitting when arrives (call this  $A_i$ )
- Thus:  $F_i = \text{MAX} (F_{i-1}, A_i) + P_i$



# FQ Algorithm (cont)

- For multiple flows
  - calculate  $F_i$  for each packet that arrives on each flow
  - treat all  $F_i$ 's as timestamps
  - next packet to transmit is one with lowest timestamp
- Not perfect: can't preempt current packet
- Example



# TCP Congestion Control

- Idea
  - assumes best-effort network (FIFO or FQ routers) each source determines network capacity for itself
  - uses implicit feedback
  - ACKs pace transmission (*self-clocking*)
- Challenge
  - determining the available capacity in the first place
  - adjusting to changes in the available capacity

# Additive Increase/Multiplicative Decrease

- Objective: adjust to changes in the available capacity
- New state variable per connection: **CongestionWindow**
  - limits how much data source has in transit

$$\text{MaxWin} = \text{MIN}(\text{CongestionWindow}, \text{AdvertisedWindow})$$
$$\text{EffWin} = \text{MaxWin} - (\text{LastByteSent} - \text{LastByteAcked})$$

- Idea:
  - increase **CongestionWindow** when congestion goes down
  - decrease **CongestionWindow** when congestion goes up

## AIMD (cont)

- Question: how does the source determine whether or not the network is congested?
- Answer: a timeout occurs
  - timeout signals that a packet was lost
  - packets are seldom lost due to transmission error
  - lost packet implies congestion

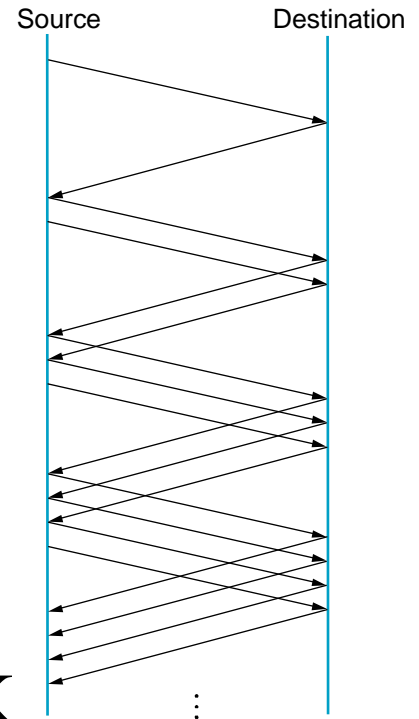
# AIMD (cont)

- Algorithm

- increment **CongestionWindow** by one packet per RTT (*linear increase*)
- divide **CongestionWindow** by two whenever a timeout occurs (*multiplicative decrease*)

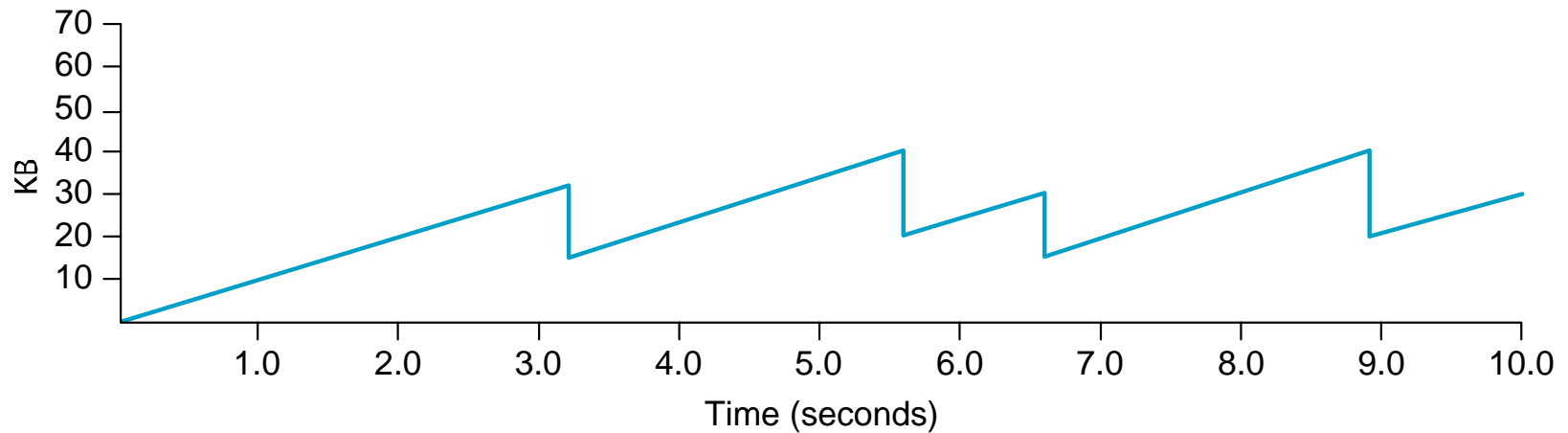
- In practice: increment a little for each ACK

**Increment** =  $(\text{MSS} * \text{MSS}) / \text{CongestionWindow}$   
**CongestionWindow** += **Increment**



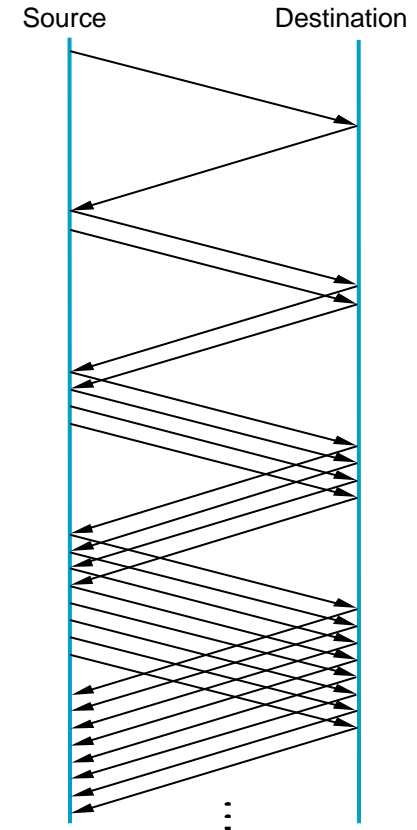
# AIMD (cont)

- Trace: sawtooth behavior



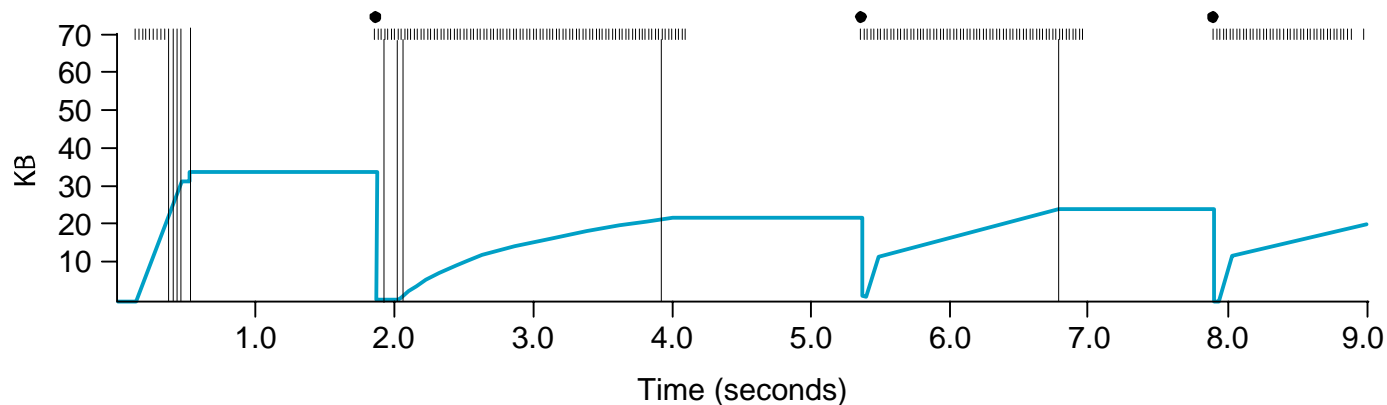
# Slow Start

- Objective: determine the available capacity in the first
- Idea:
  - begin with `CongestionWindow = 1` packet
  - double `CongestionWindow` each RTT (increment by 1 packet for each ACK)



# Slow Start (cont)

- Exponential growth, but slower than all at once
- Used...
  - when first starting connection
  - when connection goes dead waiting for timeout
- Trace

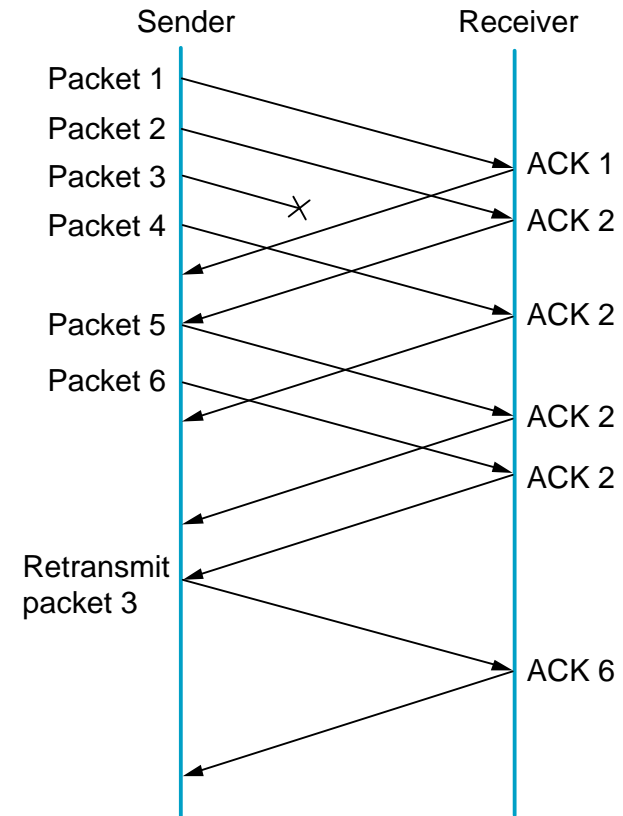


- Problem: lose up to half a **CongestionWindow**'s worth of data

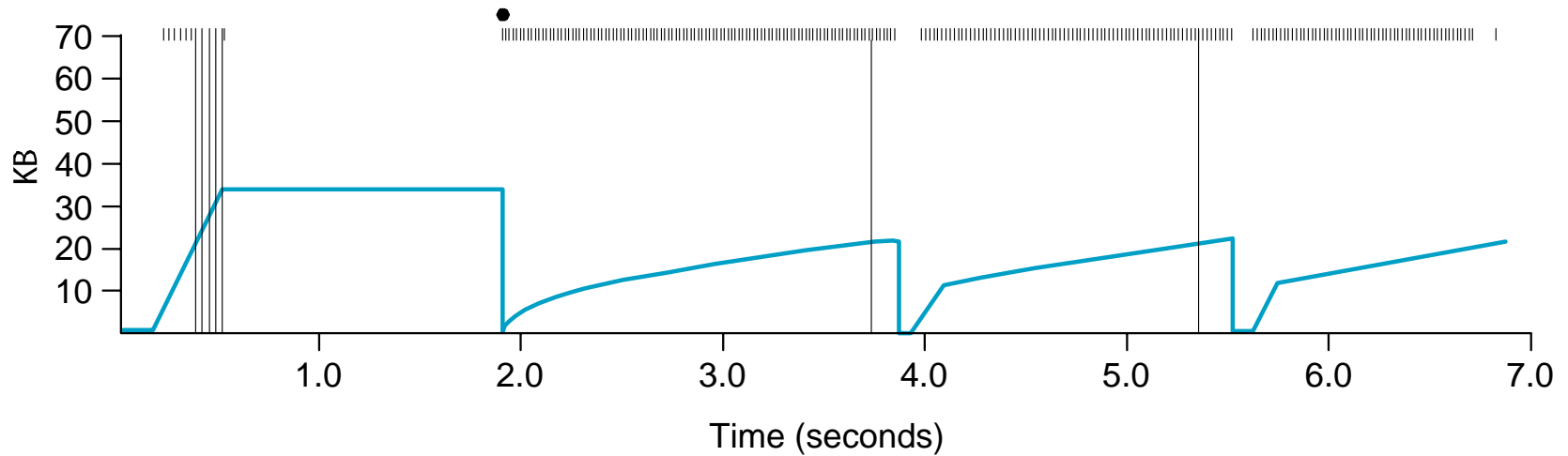


# Fast Retransmit and Fast Recovery

- Problem: coarse-grain TCP timeouts lead to idle periods
- Fast retransmit: use duplicate ACKs to trigger retransmission



# Results



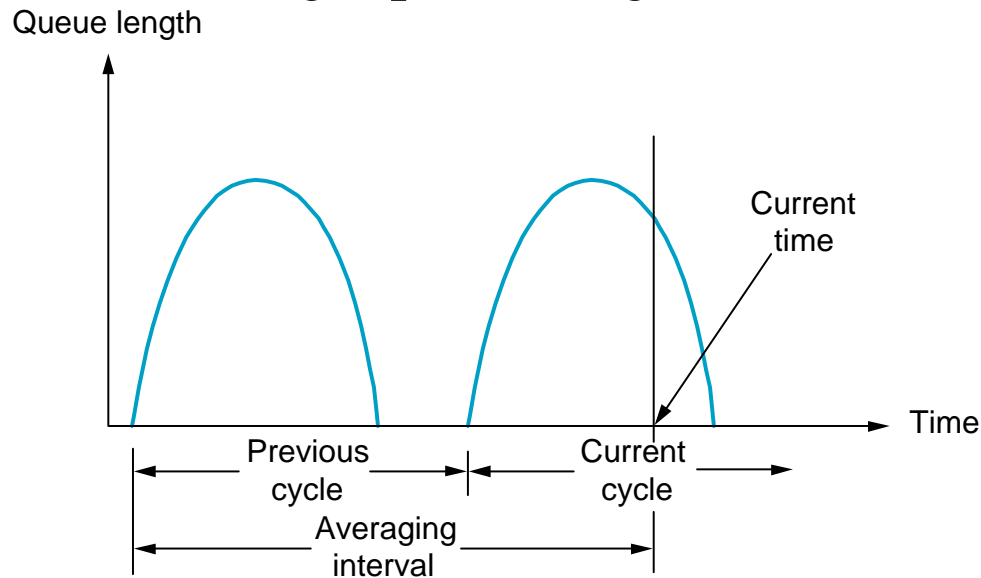
- Fast recovery
  - skip the slow start phase
  - go directly to half the last successful **CongestionWindow** (**ssthresh**)

# Congestion Avoidance

- TCP's strategy
  - control congestion once it happens
  - repeatedly increase load in an effort to find the point at which congestion occurs, and then back off
- Alternative strategy
  - predict when congestion is about to happen
  - reduce rate before packets start being discarded
  - call this congestion *avoidance*, instead of congestion *control*
- Two possibilities
  - router-centric: DECbit and RED Gateways
  - host-centric: TCP Vegas

# DECbit

- Add binary congestion bit to each packet header
- Router
  - monitors average queue length over last busy+idle cycle



- set congestion bit if average queue length  $> 1$
- attempts to balance throughput against delay

# End Hosts

- Destination echoes bit back to source
- Source records how many packets resulted in set bit
- If less than 50% of last window's worth had bit set
  - increase `CongestionWindow` by 1 packet
- If 50% or more of last window's worth had bit set
  - decrease `CongestionWindow` by 0.875 times

# Random Early Detection (RED)

- Notification is implicit
  - just drop the packet (TCP will timeout)
  - could make explicit by marking the packet
- Early random drop
  - rather than wait for queue to become full, drop each arriving packet with some *drop probability* whenever the queue length exceeds some *drop level*

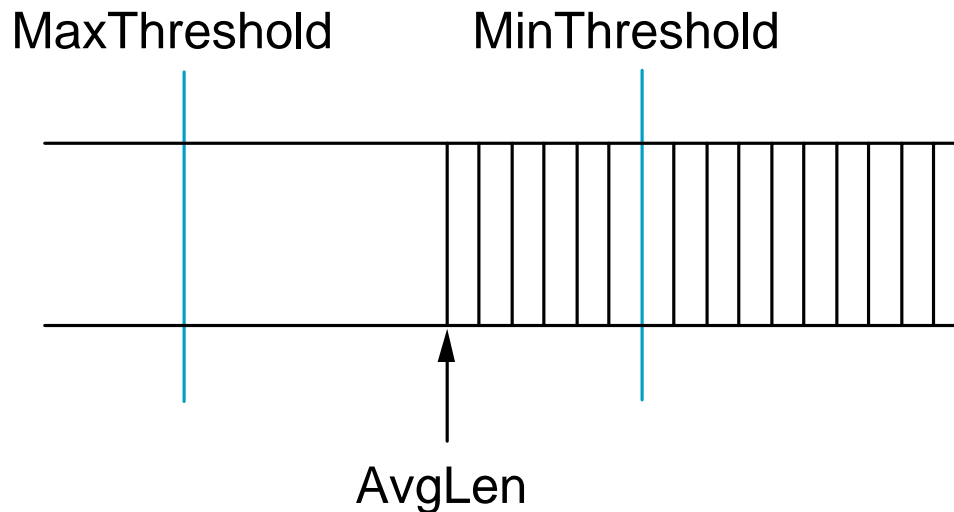
# RED Details

- Compute average queue length

$$\text{AvgLen} = (1 - \text{Weight}) * \text{AvgLen} + \text{Weight} * \text{SampleLen}$$

$0 < \text{Weight} < 1$  (usually 0.002)

**SampleLen** is queue length each time a packet arrives



# RED Details (cont)

- Two queue length thresholds

```
if AvgLen <= MinThreshold then
```

```
    enqueue the packet
```

```
if MinThreshold < AvgLen < MaxThreshold then
```

```
    calculate probability P
```

```
    drop arriving packet with probability P
```

```
if MaxThreshold <= AvgLen then
```

```
    drop arriving packet
```



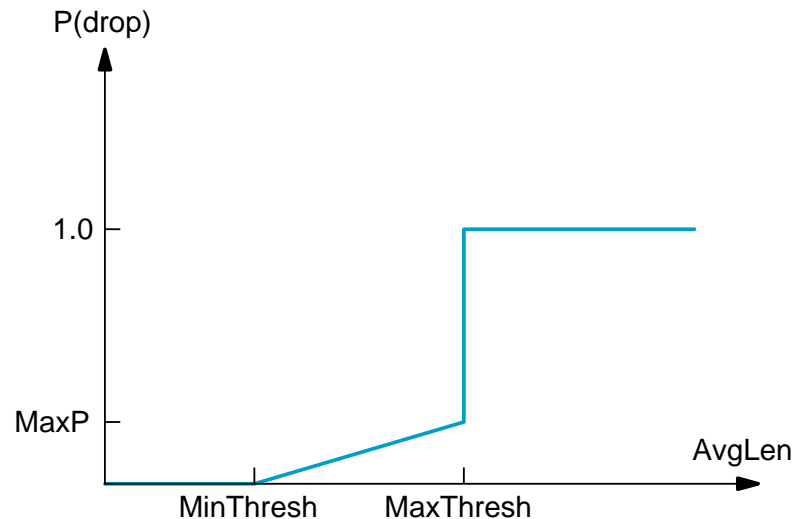
# RED Details (cont)

- Computing probability P

$$\text{TempP} = \text{MaxP} * (\text{AvgLen} - \text{MinThreshold}) / (\text{MaxThreshold} - \text{MinThreshold})$$

$$P = \text{TempP} / (1 - \text{count} * \text{TempP})$$

- Drop Probability Curve

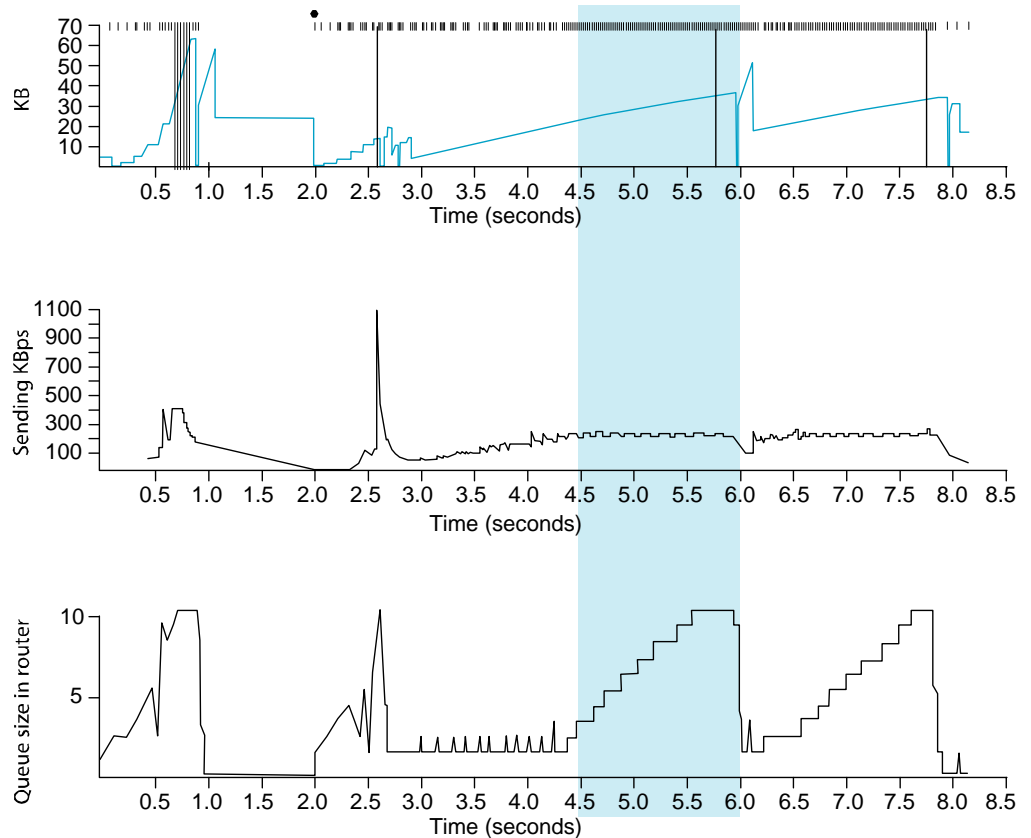


# Tuning RED

- Probability of dropping a particular flow's packet(s) is roughly proportional to the share of the bandwidth that flow is currently getting
- **MaxP** is typically set to 0.02, meaning that when the average queue size is halfway between the two thresholds, the gateway drops roughly one out of 50 packets.
- If traffic is bursty, then **MinThreshold** should be sufficiently large to allow link utilization to be maintained at an acceptably high level
- Difference between two thresholds should be larger than the typical increase in the calculated average queue length in one RTT; setting **MaxThreshold** to twice **MinThreshold** is reasonable for traffic on today's Internet
- Penalty Box for Offenders

# TCP Vegas

- Idea: source watches for some sign that router's queue is building up and congestion will happen too; e.g.,
  - RTT grows
  - sending rate flattens



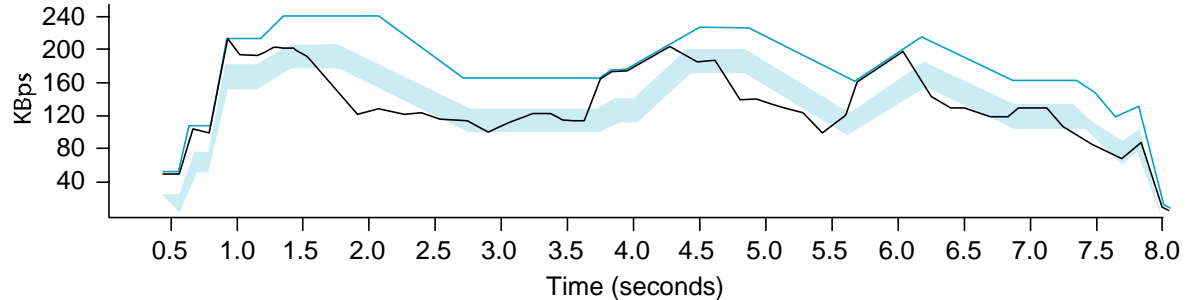
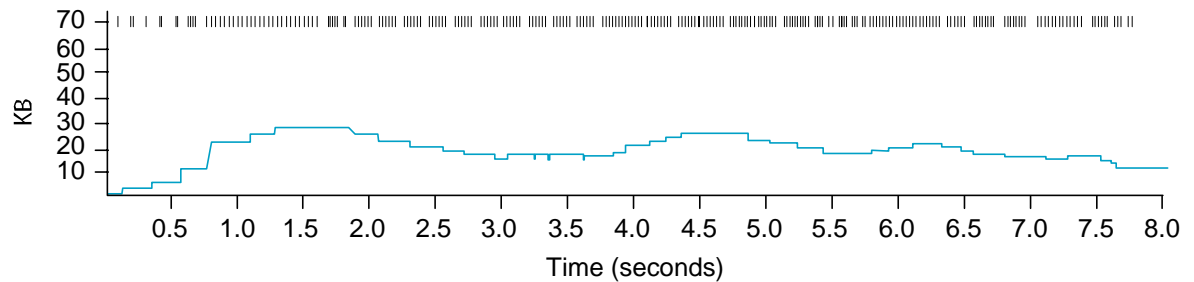
# Algorithm

- Let **BaseRTT** be the minimum of all measured RTTs (commonly the RTT of the first packet)
- If not overflowing the connection, then  
     $\text{ExpectRate} = \text{CongestionWindow} / \text{BaseRTT}$
- Source calculates sending rate (**ActualRate**) once per RTT
- Source compares **ActualRate** with **ExpectRate**

```
Diff = ExpectedRate - ActualRate
if Diff <  $\alpha$ 
    increase CongestionWindow linearly
else if Diff >  $\beta$ 
    decrease CongestionWindow linearly
else
    leave CongestionWindow unchanged
```

# Algorithm (cont)

- Parameters
  - $\alpha = 1$  packet
  - $\beta = 3$  packets



- Even faster retransmit
  - keep fine-grained timestamps for each packet
  - check for timeout on first duplicate ACK

# *Quality of Service*

*Go To Talk Outline*

# Quality of Service

## Outline

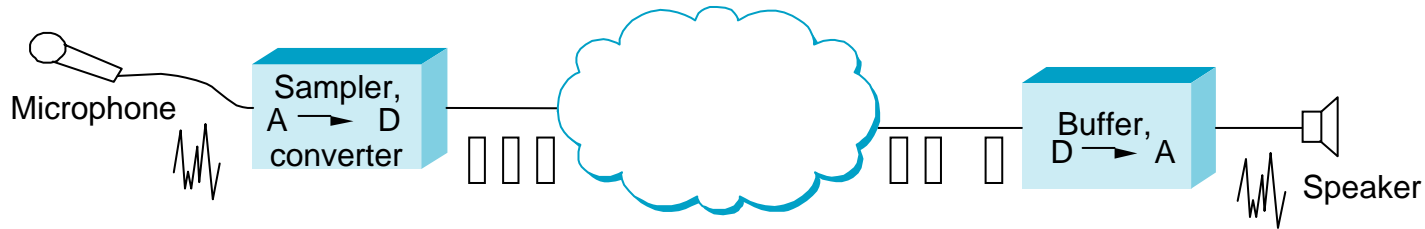
Realtime Applications

Integrated Services

Differentiated Services

# Realtime Applications

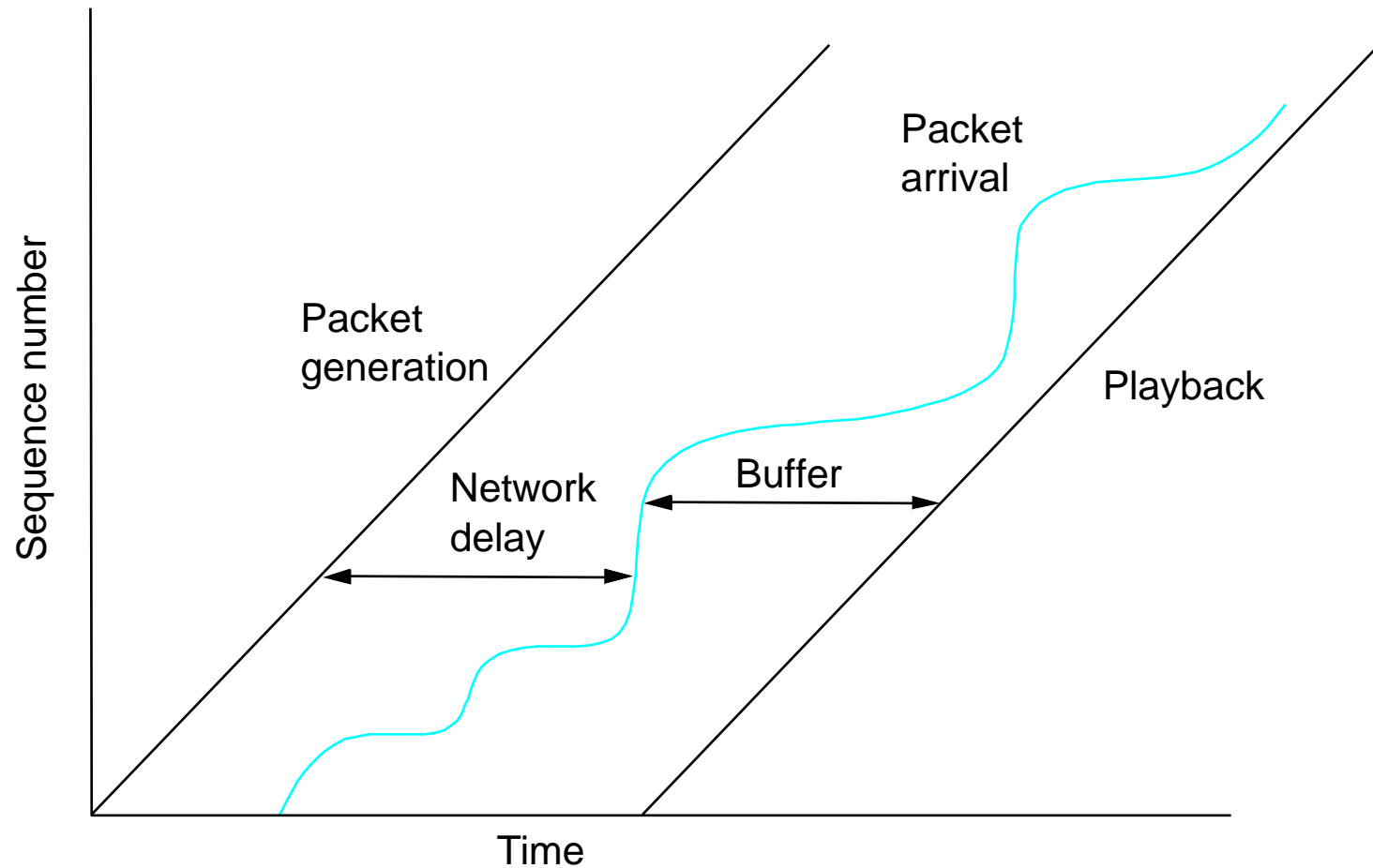
- Require “deliver on time” assurances
  - must come from *inside* the network



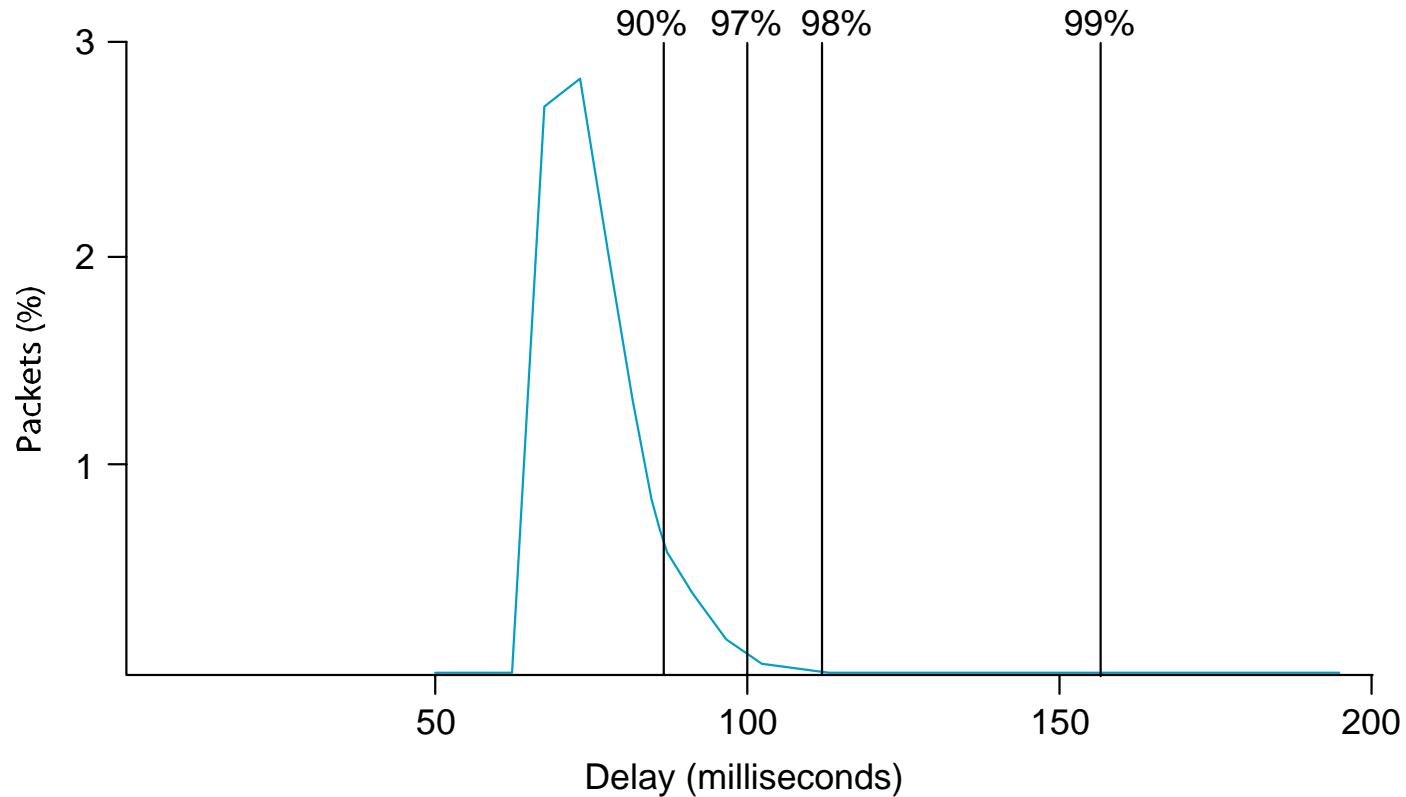
- Example application (audio)
  - sample voice once every 125us
  - each sample has a *playback time*
  - packets experience variable delay in network
  - add constant factor to playback time: *playback point*



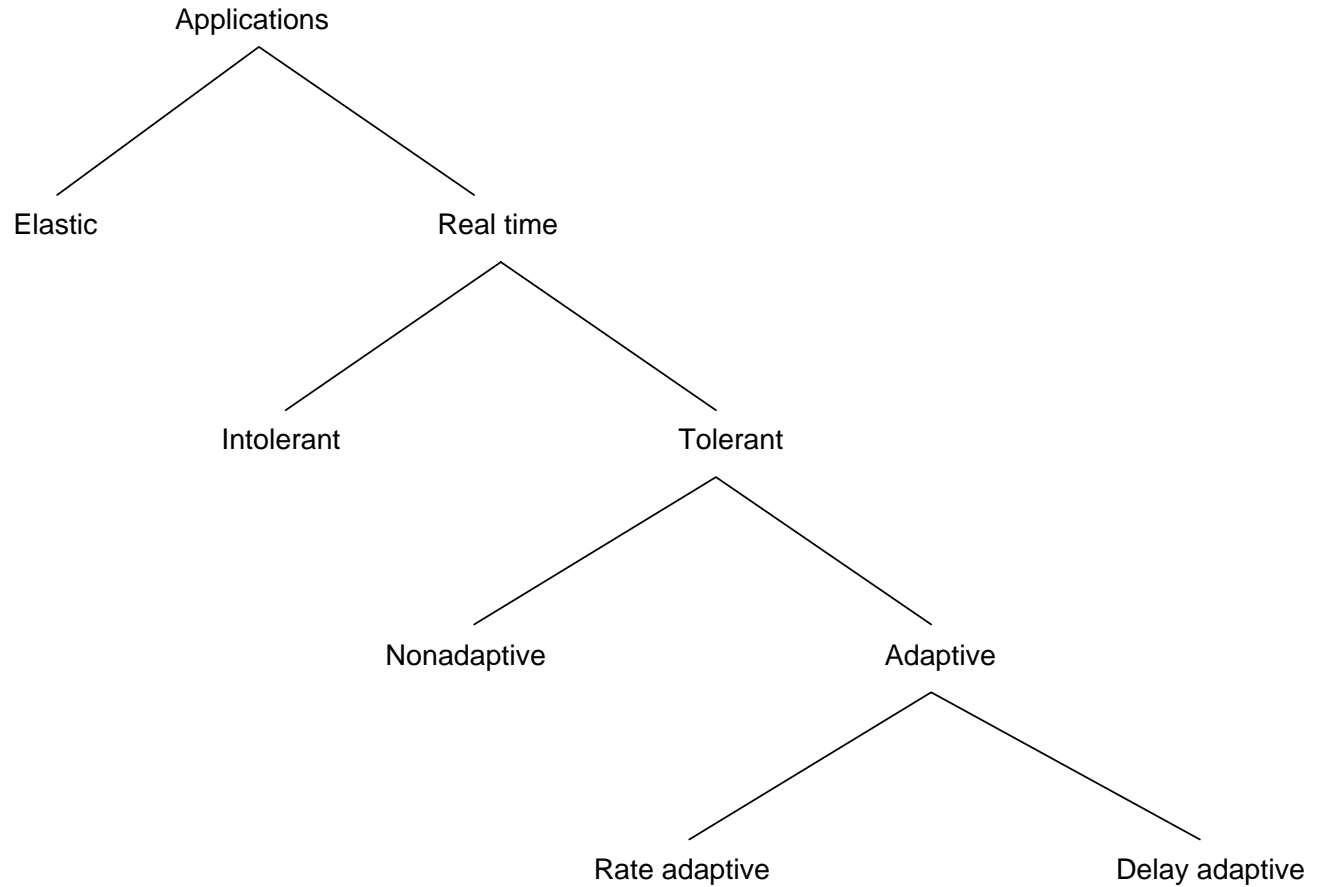
# Playback Buffer



# Example Distribution of Delays



# Taxonomy



# Integrated Services

- Service Classes
  - guaranteed
  - controlled-load
- Mechanisms
  - signalling protocol
  - admission control
  - policing
  - packet scheduling

# Flowspec

- ***Rspec***: describes service requested from network
  - controlled-load: none
  - guaranteed: delay target
- ***Tspec***: describes flow's traffic characteristics
  - average bandwidth + burstiness: *token bucket* filter
  - token rate  $r$
  - bucket depth  $B$
  - must have a token to send a byte
  - must have  $n$  tokens to send  $n$  bytes
  - start with no tokens
  - accumulate tokens at rate of  $r$  per second
  - can accumulate no more than  $B$  tokens

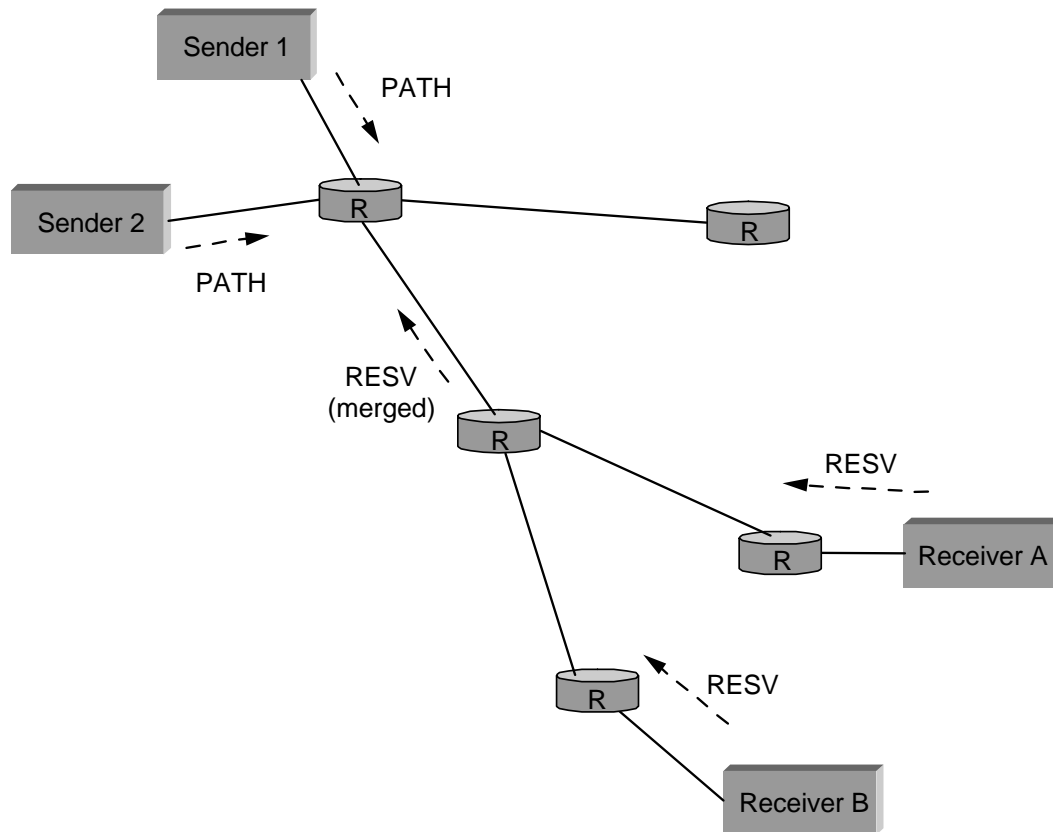
# Per-Router Mechanisms

- Admission Control
  - decide if a new flow can be supported
  - answer depends on service class
  - not the same as *policing*
- Packet Processing
  - classification: associate each packet with the appropriate reservation
  - scheduling: manage queues so each packet receives the requested service

# Reservation Protocol

- Called signaling in ATM
- Proposed Internet standard: RSVP
- Consistent with robustness of today's connectionless model
- Uses soft state (refresh periodically)
- Designed to support multicast
- Receiver-oriented
- Two messages: PATH and RESV
- Source transmits PATH messages every 30 seconds
- Destination responds with RESV message
- Merge requirements in case of multicast
- Can specify number of speakers

# RSVP Example





# RSVP versus ATM (Q.2931)

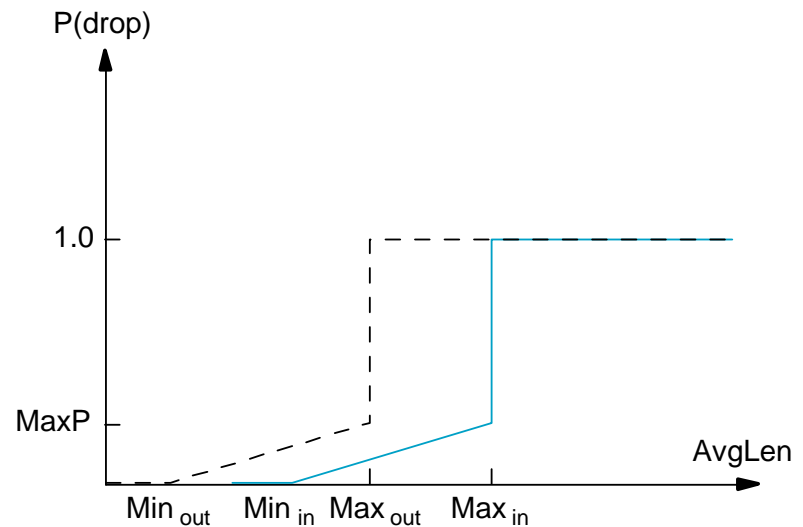
- RSVP
  - receiver generates reservation
  - soft state (refresh/timeout)
  - separate from route establishment
  - QoS can change dynamically
  - receiver heterogeneity
- ATM
  - sender generates connection request
  - hard state (explicit delete)
  - concurrent with route establishment
  - QoS is static for life of connection
  - uniform QoS to all receivers

# Differentiated Services

- Problem with IntServ: scalability
- Idea: segregate packets into a small number of classes
  - e.g., premium vs best-effort
- Packets marked according to class at edge of network
- Core routers implement some per-hop-behavior (PHB)
- Example: Expedited Forwarding (EF)
  - rate-limit EF packets at the edges
  - PHB implemented with class-based priority queues or WFQ

# DiffServ (cont)

- Assured Forwarding (AF)
  - customers sign service agreements with ISPs
  - edge routers mark packets as being “in” or “out” of profile
  - core routers run RIO: RED with in/out



## *Remote Procedure Call*

*Go To Talk Outline*

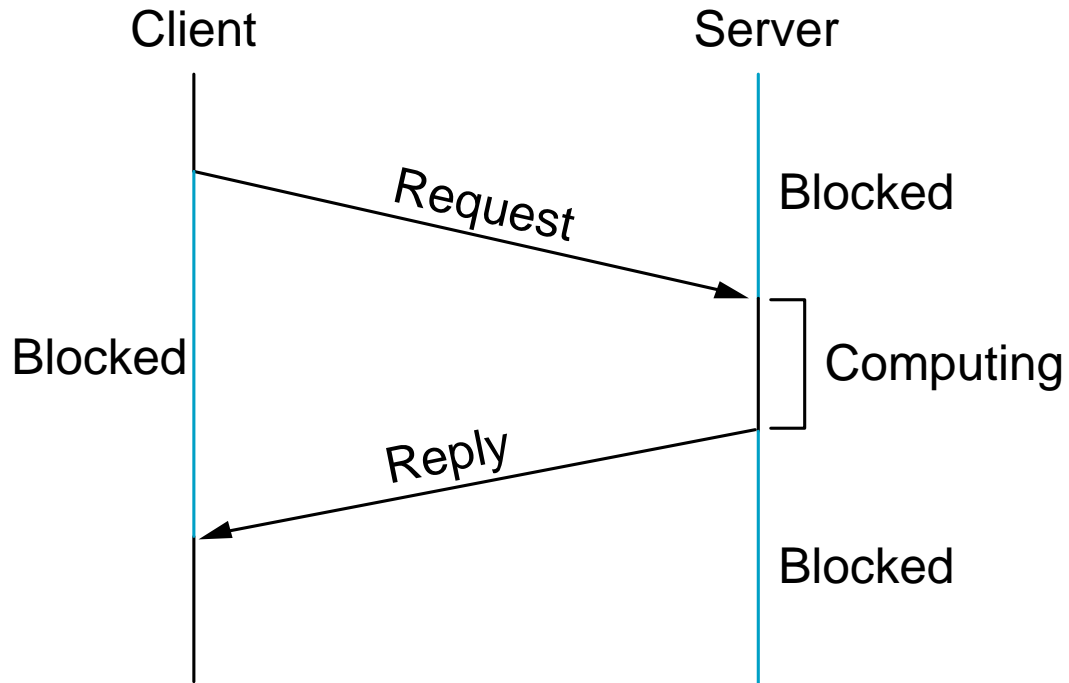
# Remote Procedure Call

## Outline

Protocol Stack

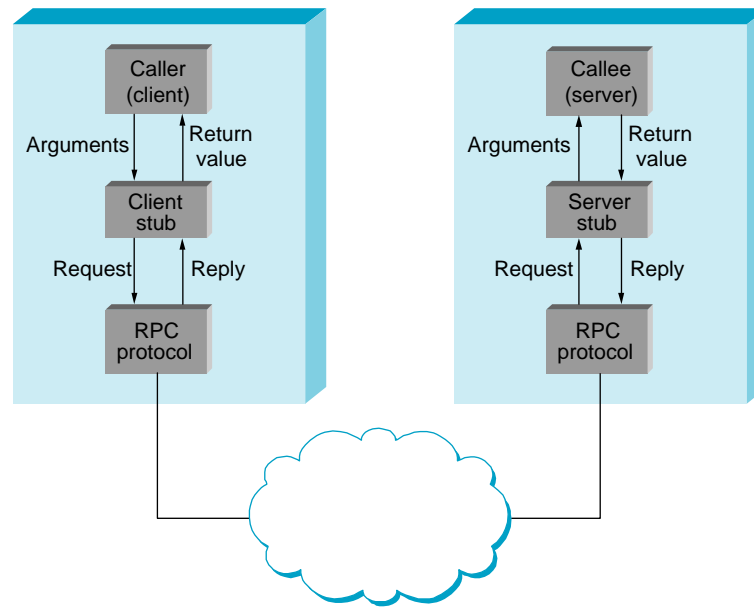
Presentation Formatting

# RPC Timeline



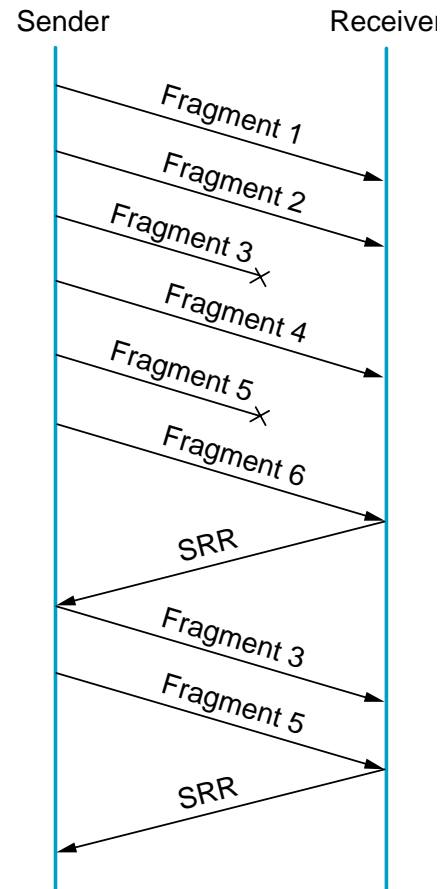
# RCP Components

- Protocol Stack
  - BLAST: fragments and reassembles large messages
  - CHAN: synchronizes request and reply messages
  - SELECT: dispatches request to the correct process
- Stubs



# Bulk Transfer (BLAST)

- Unlike AAL and IP, tries to recover from lost fragments
- Strategy
  - selective retransmission
  - aka partial acknowledgements





# BLAST Details

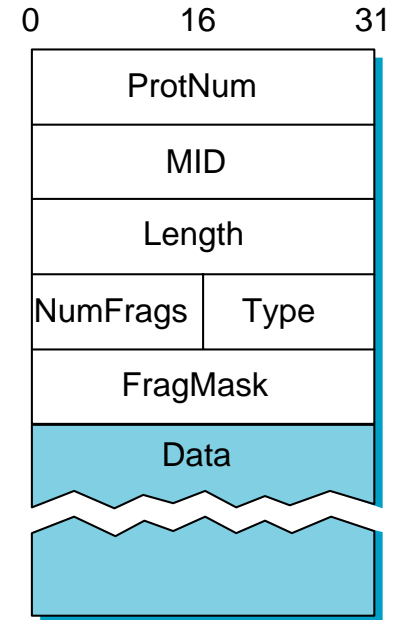
- Sender:
  - after sending all fragments, set timer DONE
  - if receive SRR, send missing fragments and reset DONE
  - if timer DONE expires, free fragments

## BLAST Details (cont)

- Receiver:
  - when first fragments arrives, set timer LAST\_FRAG
  - when all fragments present, reassemble and pass up
  - four exceptional conditions:
    - if last fragment arrives but message not complete
      - send SRR and set timer RETRY
    - if timer LAST\_FRAG expires
      - send SRR and set timer RETRY
    - if timer RETRY expires for first or second time
      - send SRR and set timer RETRY
    - if timer RETRY expires a third time
      - give up and free partial message

# BLAST Header Format

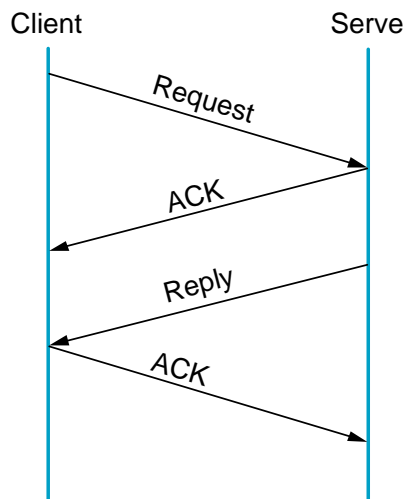
- MID must protect against wrap around
- TYPE = DATA or SRR
- NumFrag indicates number of fragments
- FragMask distinguishes among fragments
  - if Type=DATA, identifies this fragment
  - if Type=SRR, identifies missing fragments



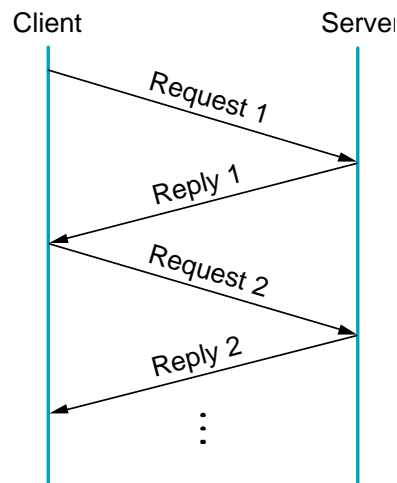
# Request/Reply (CHAN)

- Guarantees message delivery
- Synchronizes client with server
- Supports *at-most-once* semantics

## Simple case



## Implicit Acks



# CHAN Details

- Lost message (request, reply, or ACK)
  - set RETRANSMIT timer
  - use message id (MID) field to distinguish
- Slow (long running) server
  - client periodically sends “are you alive” probe, or
  - server periodically sends “I’m alive” notice
- Want to support multiple outstanding calls
  - use channel id (CID) field to distinguish
- Machines crash and reboot
  - use boot id (BID) field to distinguish

# CHAN Header Format

```
typedef struct {  
    u_short  Type;      /* REQ, REP, ACK, PROBE */  
    u_short  CID;       /* unique channel id */  
    int      MID;       /* unique message id */  
    int      BID;       /* unique boot id */  
    int      Length;    /* length of message */  
    int      ProtNum;   /* high-level protocol */  
} ChanHdr;
```

```
typedef struct {  
    u_char    type;      /* CLIENT or SERVER */  
    u_char    status;    /* BUSY or IDLE */  
    int       retries;   /* number of retries */  
    int       timeout;   /* timeout value */  
    XkReturn  ret_val;   /* return value */  
    Msg       *request;  /* request message */  
    Msg       *reply;    /* reply message */  
    Semaphore reply_sem; /* client semaphore */  
    int       mid;       /* message id */  
    int       bid;       /* boot id */  
} ChanState;
```

# Synchronous vs Asynchronous Protocols

- Asynchronous interface

`send(Protocol llp, Msg *message)`

`deliver(Protocol llp, Msg *message)`

- Synchronous interface

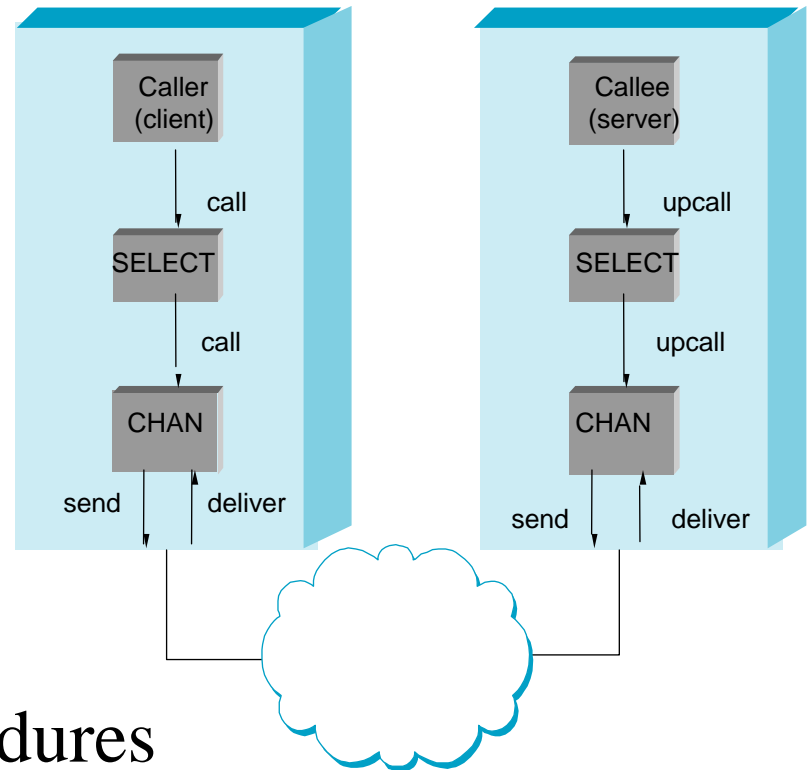
`call(Protocol llp, Msg *request, Msg *reply)`

`upcall(Protocol hlp, Msg *request, Msg *reply)`

- CHAN is a hybrid protocol
  - synchronous from above: **call**
  - asynchronous from below: **deliver**

# Dispatcher (SELECT)

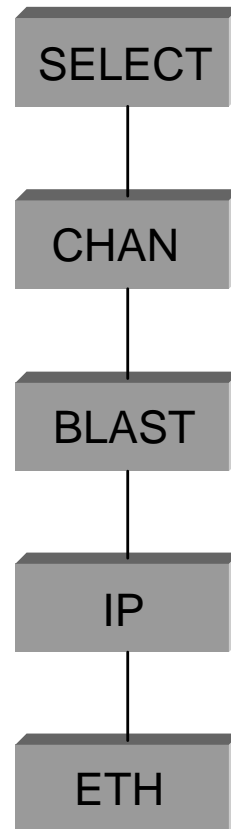
- Dispatch to appropriate procedure
- Synchronous counterpart to UDP
- Implement concurrency (open multiple CHANs)



- Address Space for Procedures
  - flat: unique id for each possible procedure
  - hierarchical: program + procedure number

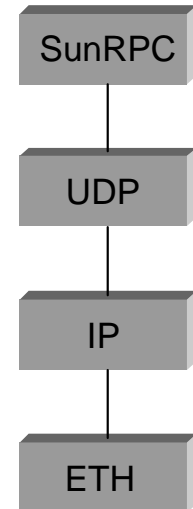


# Simple RPC Stack



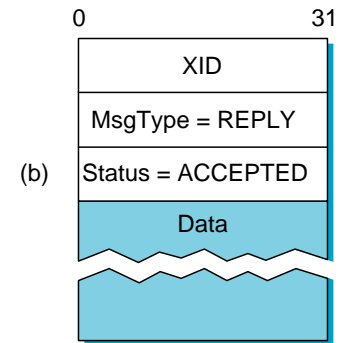
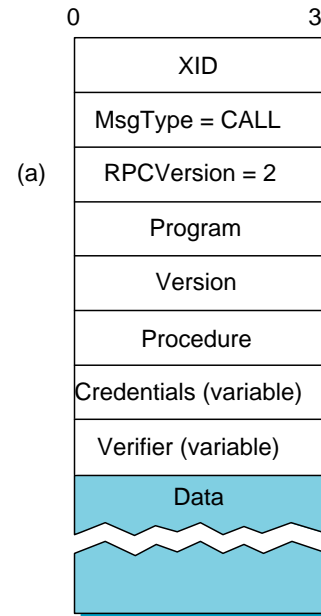
# SunRPC

- IP implements BLAST-equivalent
  - except no selective retransmit
- SunRPC implements CHAN-equivalent
  - except not at-most-once
- UDP + SunRPC implement SELECT-equivalent
  - UDP dispatches to program (ports bound to programs)
  - SunRPC dispatches to procedure within program



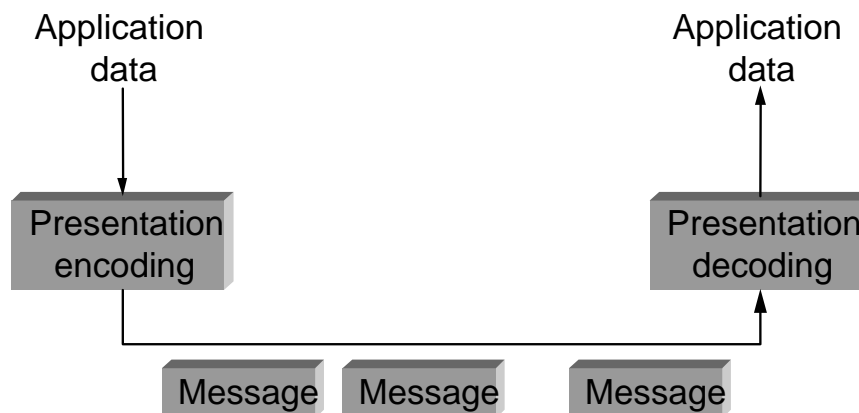
# SunRPC Header Format

- XID (transaction id) is similar to CHAN's MID
- Server does not remember last XID it serviced
- Problem if client retransmits request while reply is in transit



# Presentation Formatting

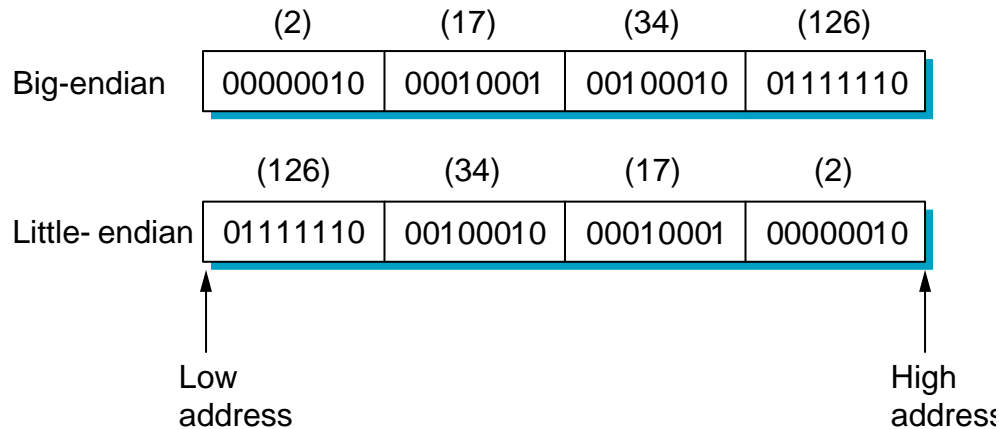
- Marshalling (encoding) application data into messages
- Unmarshalling (decoding) messages into application data



- Data types we consider
  - integers
  - floats
  - strings
  - arrays
  - structs
- Types of data we do not consider
  - images
  - video
  - multimedia documents

# Difficulties

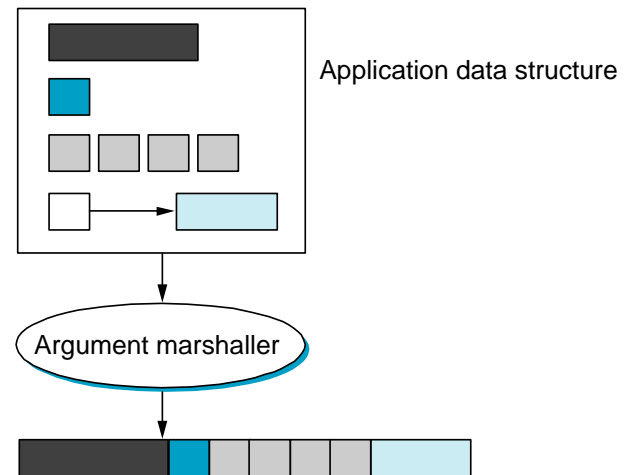
- Representation of base types
  - floating point: IEEE 754 versus non-standard
  - integer: big-endian versus little-endian (e.g., 34,677,374)



- Compiler layout of structures

# Taxonomy

- Data types
  - base types (e.g., ints, floats); must convert
  - flat types (e.g., structures, arrays); must pack
  - complex types (e.g., pointers); must linearize



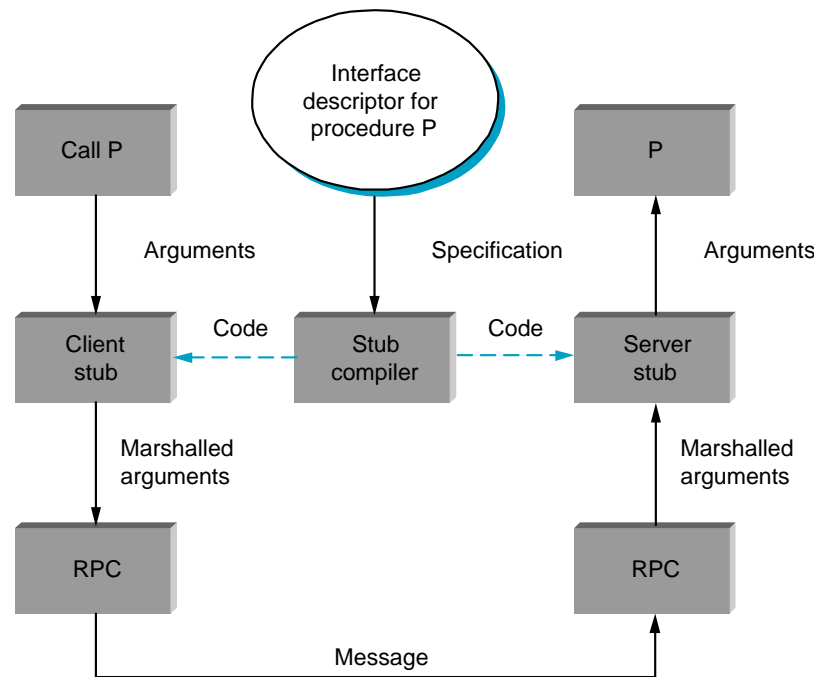
- Conversion Strategy
  - canonical intermediate form
  - receiver-makes-right (an  $N \times N$  solution)

# Taxonomy (cont)

- Tagged versus untagged data

type = INT	len = 4		value = 417892	
---------------	---------	--	----------------	--

- Stubs
  - compiled
  - interpreted



# eXternal Data Representation (XDR)

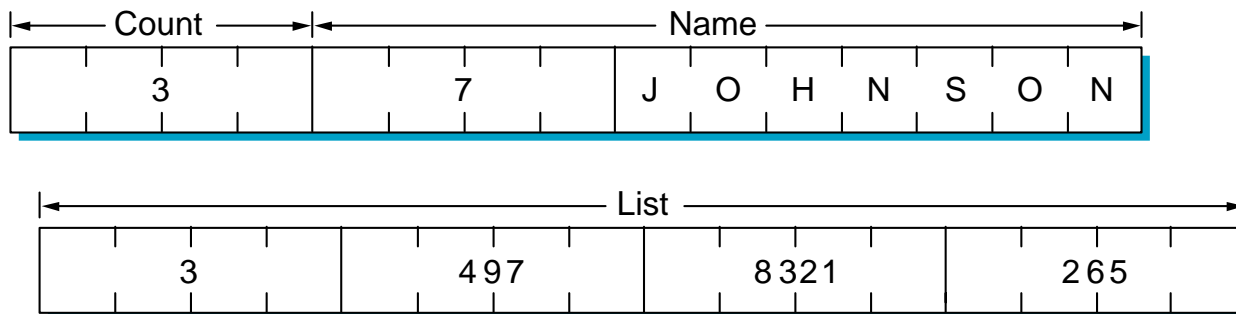
- Defined by Sun for use with SunRPC
- C type system (without function pointers)
- Canonical intermediate form
- Untagged (except array length)
- Compiled stubs



```
#define MAXNAME 256;
#define MAXLIST 100;
```

```
struct item {
    int    count;
    char    name[MAXNAME];
    int    list[MAXLIST];
};
```

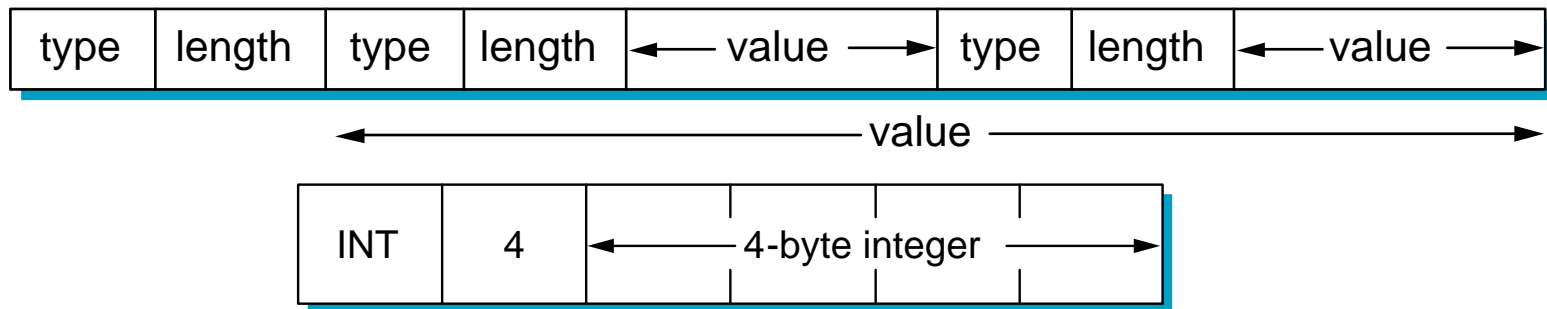
```
bool_t
xdr_item(XDR *xdrs, struct item *ptr)
{
    return(xdr_int(xdrs, &ptr->count) &&
           xdr_string(xdrs, &ptr->name, MAXNAME) &&
           xdr_array(xdrs, &ptr->list, &ptr->count,
                     MAXLIST, sizeof(int), xdr_int));
}
```



# Abstract Syntax Notation One (ASN-1)

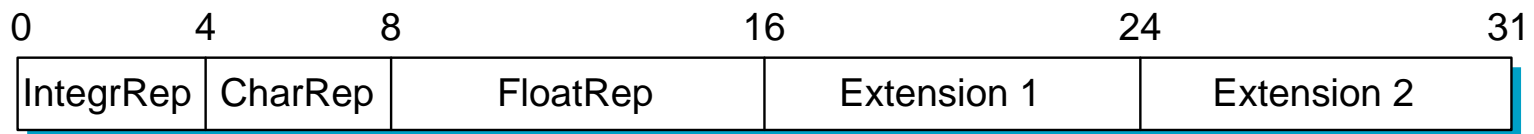
- An ISO standard
- Essentially the C type system
- Canonical intermediate form
- Tagged
- Compiled or interpreted stubs
- BER: Basic Encoding Rules

**(tag, length, value)**



# Network Data Representation (NDR)

- Defined by DCE
  - Essentially the C type system
  - Receiver-makes-right (architecture tag)
  - Individual data items untagged
  - Compiled stubs from IDL
  - 4-byte architecture tag
- IntegerRep
    - 0 = big-endian
    - 1 = little-endian
  - CharRep
    - 0 = ASCII
    - 1 = EBCDIC
  - FloatRep
    - 0 = IEEE 754
    - 1 = VAX
    - 2 = Cray
    - 3 = IBM



# *Naming and Security*

*Go To Talk Outline*

# Naming

## Outline

Terminology

Domain Naming System

Distributed File Systems

# Overview

- What do names do?
  - identify objects
  - help locate objects
  - define membership in a group
  - specify a role
  - convey knowledge of a secret
- Name space
  - defines set of possible names
  - consists of a set of name to value *bindings*

# Properties

- Names versus addresses
- Location transparent versus location-dependent
- Flat versus hierarchical
- Global versus local
- Absolute versus relative
- By architecture versus by convention
- Unique versus ambiguous

# Examples

- Hosts

`cheltenham.cs.princeton.edu`  $\longrightarrow$  `192.12.69.17`

`192.12.69.17`  $\longrightarrow$  `80:23:A8:33:5B:9F`

- Files

`/usr/llp/tmp/foo`  $\longrightarrow$  `(server, fileid)`

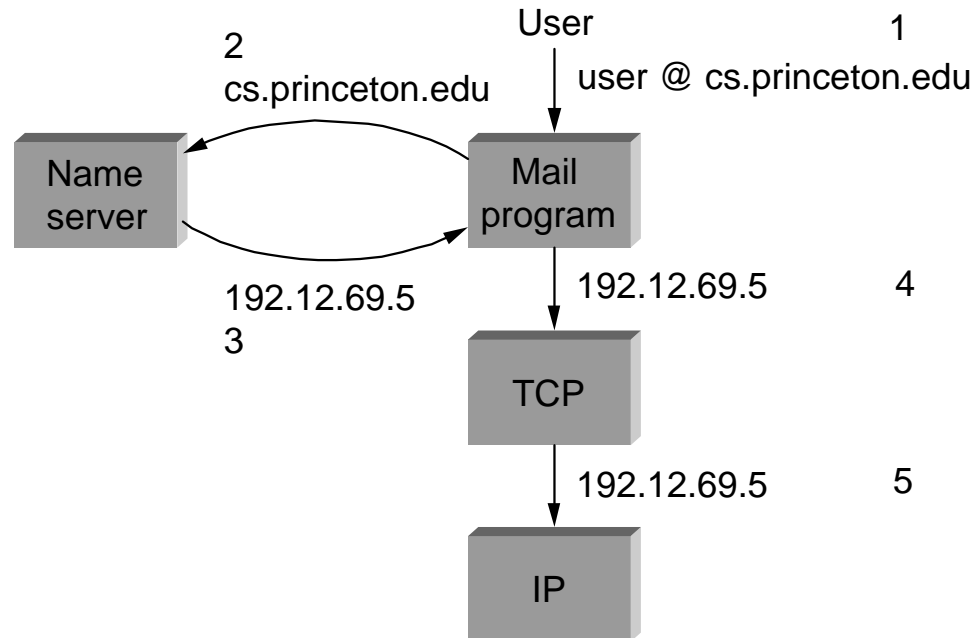
- Users

`Larry Peterson`  $\longrightarrow$  `llp@cs.princeton.edu`



# Examples (cont)

- Mailboxes

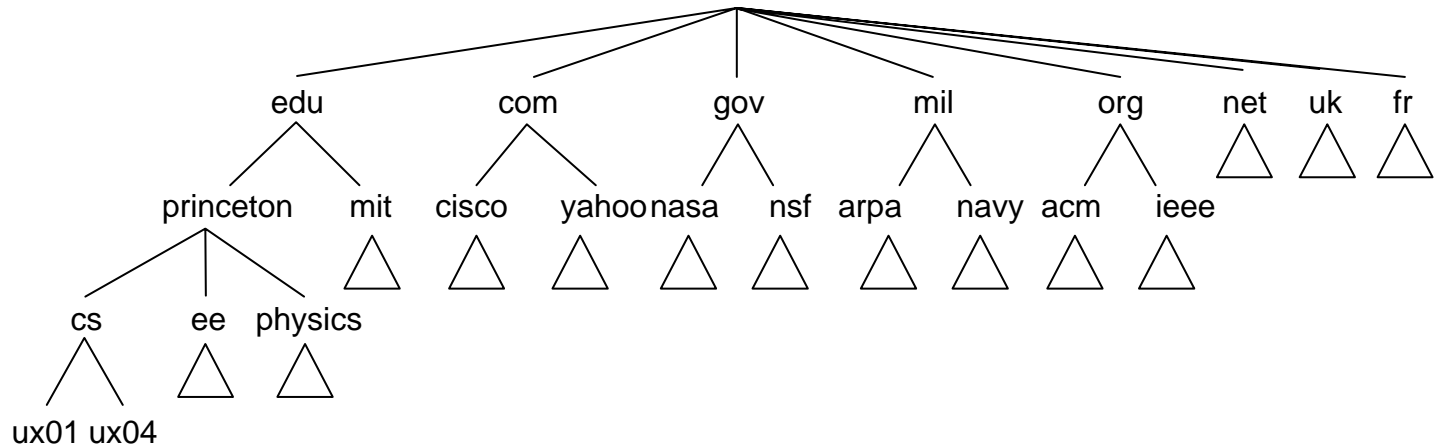


- Services

**nearby ps printer with short queue and 2MB**

# Domain Naming System

- Hierarchy

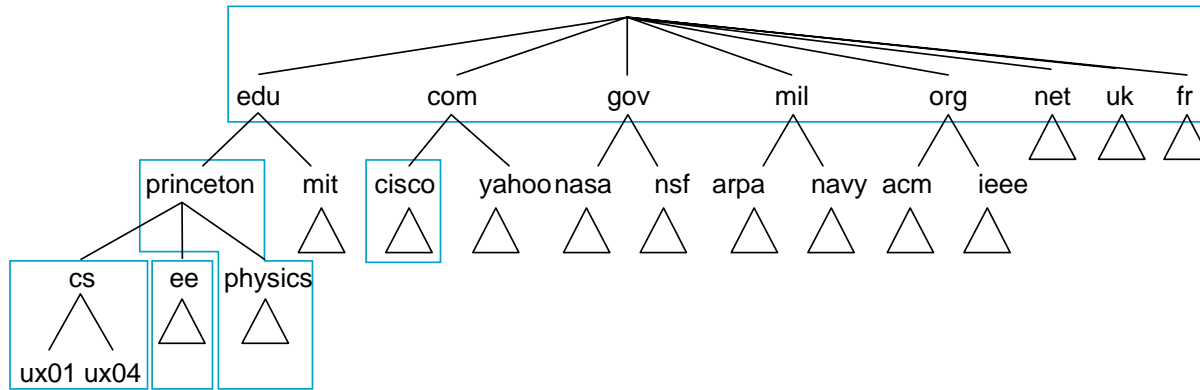


- Name

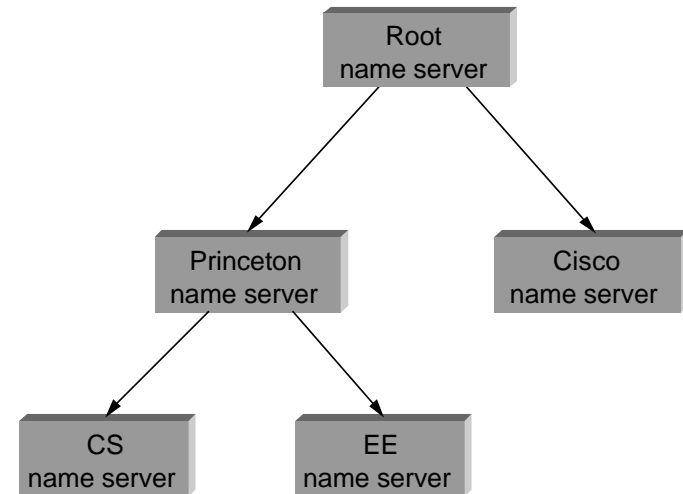
**chinstrap.cs.princeton.edu**

# Name Servers

- Partition hierarchy into *zones*



- Each zone implemented by two or more *name servers*



# Resource Records

- Each name server maintains a collection of *resource records*  
(Name, Value, Type, Class, TTL)
- Name/Value: not necessarily host names to IP addresses
- Type
  - NS: Value gives domain name for host running name server that knows how to resolve names within specified domain.
  - CNAME: Value gives canonical name for particle host; used to define aliases.
  - MX: Value gives domain name for host running mail server that accepts messages for specified domain.
- Class: allow other entities to define types
- TTL: how long the resource record is valid

# Root Server

`(princeton.edu, cit.princeton.edu, NS, IN)`  
`(cit.princeton.edu, 128.196.128.233, A, IN)`

`(cisco.com, thumper.cisco.com, NS, IN)`  
`(thumper.cisco.com, 128.96.32.20, A, IN)`

...

# Princeton Server

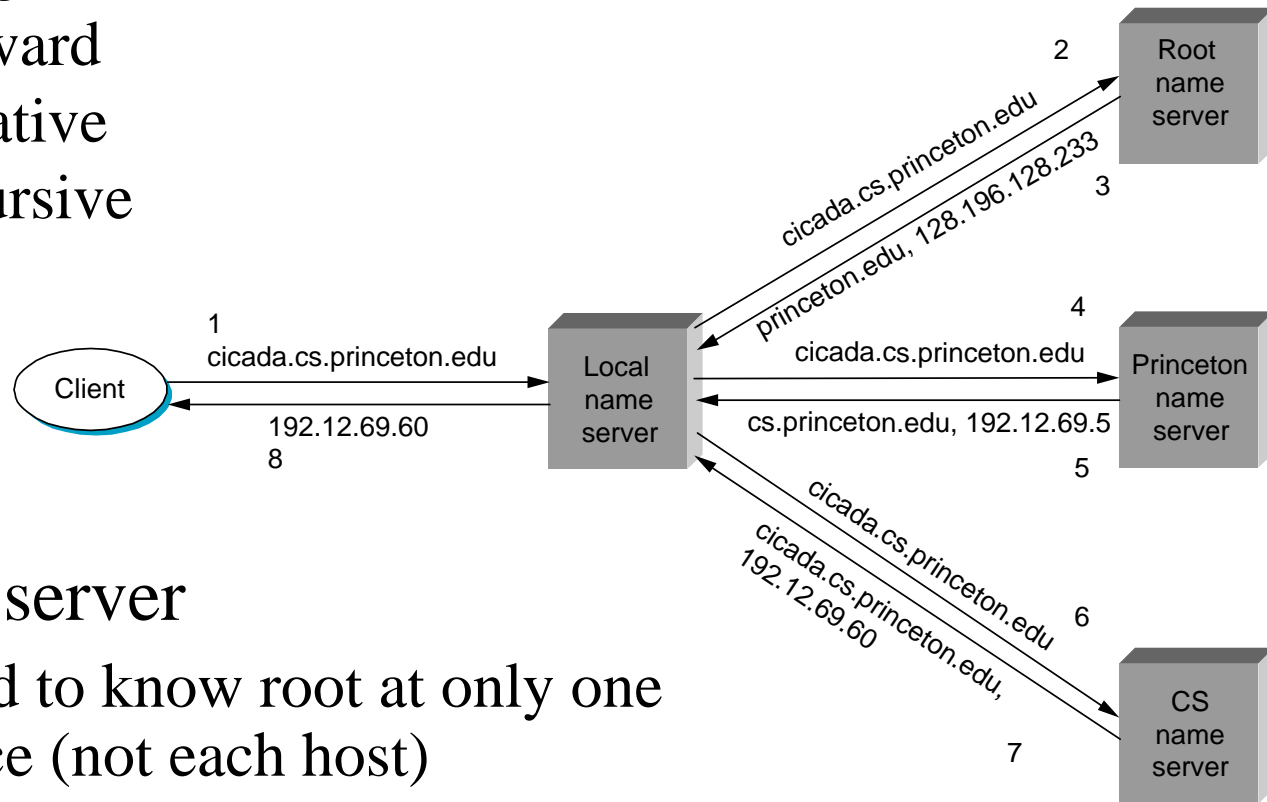
```
(cs.princeton.edu, optima.cs.princeton.edu, NS, IN)
(optima.cs.princeton.edu, 192.12.69.5, A, IN)
(ee.princeton.edu, helios.ee.princeton.edu, NS, IN)
(helios.ee.princeton.edu, 128.196.28.166, A, IN)
(jupiter.physics.princeton.edu, 128.196.4.1, A, IN)
(saturn.physics.princeton.edu, 128.196.4.2, A, IN)
(mars.physics.princeton.edu, 128.196.4.3, A, IN)
(venus.physics.princeton.edu, 128.196.4.4, A, IN)
```

# CS Server

```
(cs.princeton.edu, optima.cs.princeton.edu, MX, IN)
(cheltenham.cs.princeton.edu, 192.12.69.60, A, IN)
(che.cs.princeton.edu, cheltenham.cs.princeton.edu,
 CNAME, IN)
(optima.cs.princeton.edu, 192.12.69.5, A, IN)
(opt.cs.princeton.edu, optima.cs.princeton.edu,
 CNAME, IN)
(baskerville.cs.princeton.edu, 192.12.69.35, A, IN)
(bas.cs.princeton.edu, baskerville.cs.princeton.edu,
 CNAME, IN)
```

# Name Resolution

- Strategies
  - forward
  - iterative
  - recursive



- Local server
  - need to know root at only one place (not each host)
  - site-wide cache



# Distributed File Systems

- No Transparency

Global AFS: `/cs.princeton.edu/usr/llp/tmp/foo`

Windows: `f:/usr/llp/tmp/foo`

- Transparency by Convention

- NFS: `/usr/llp/tmp/foo`

- Or Not: `/n/fs/fac5/llp/tmp/foo`

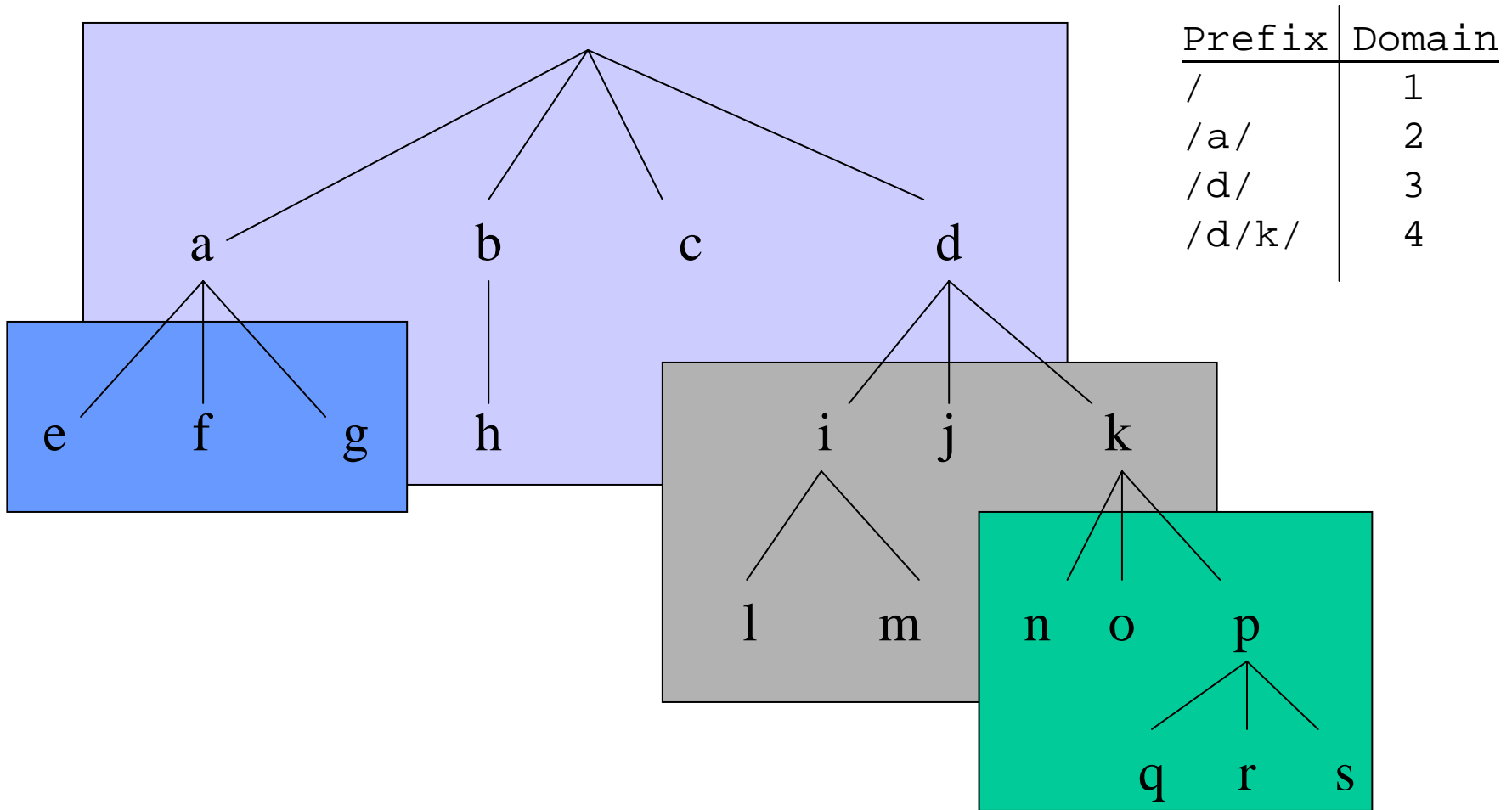
- Transparency by Architecture

- Sprite: `/usr/llp/tmp/foo`

- Private versus Shared

- ASF: `/usr/llp/tmp/foo` versus `/afs/shared`

# Example



# Stupid Naming Tricks

- Symbolic links and mount points
- Per-User and logical name spaces
- Computed directories
- Load balancing and content distribution
- Attribute-based names
- Hash-based schemes

# Security

## Outline

Encryption Algorithms

Authentication Protocols

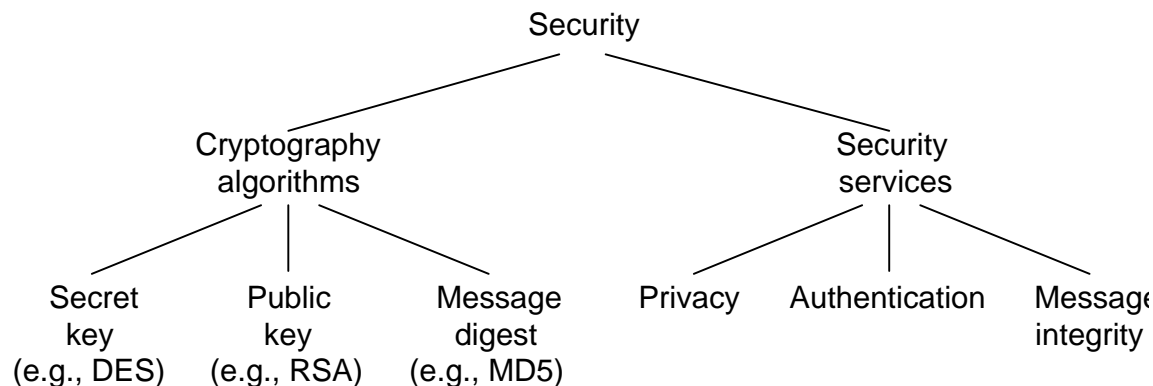
Message Integrity Protocols

Key Distribution

Firewalls

# Overview

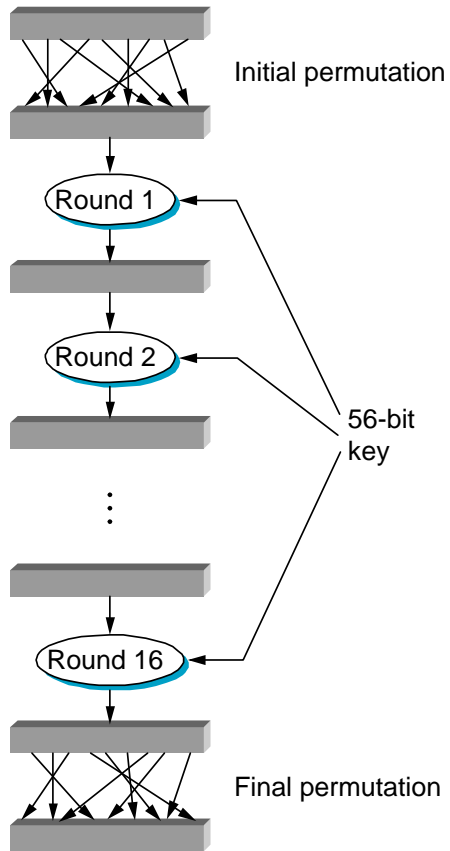
- Cryptography functions
  - Secret key (e.g., DES)
  - Public key (e.g., RSA)
  - Message digest (e.g., MD5)
- Security services
  - Privacy: preventing unauthorized release of information
  - Authentication: verifying identity of the remote participant
  - Integrity: making sure message has not been altered



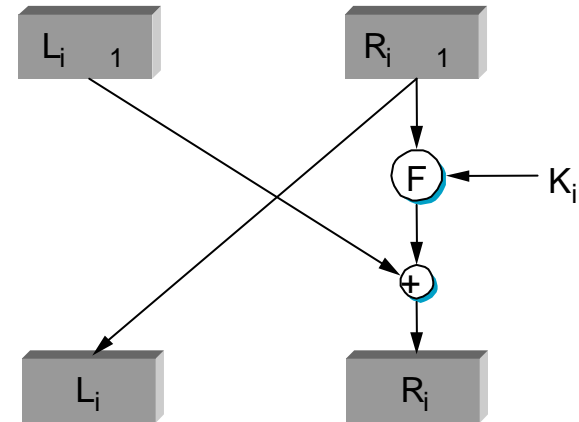
# Secret Key (DES)



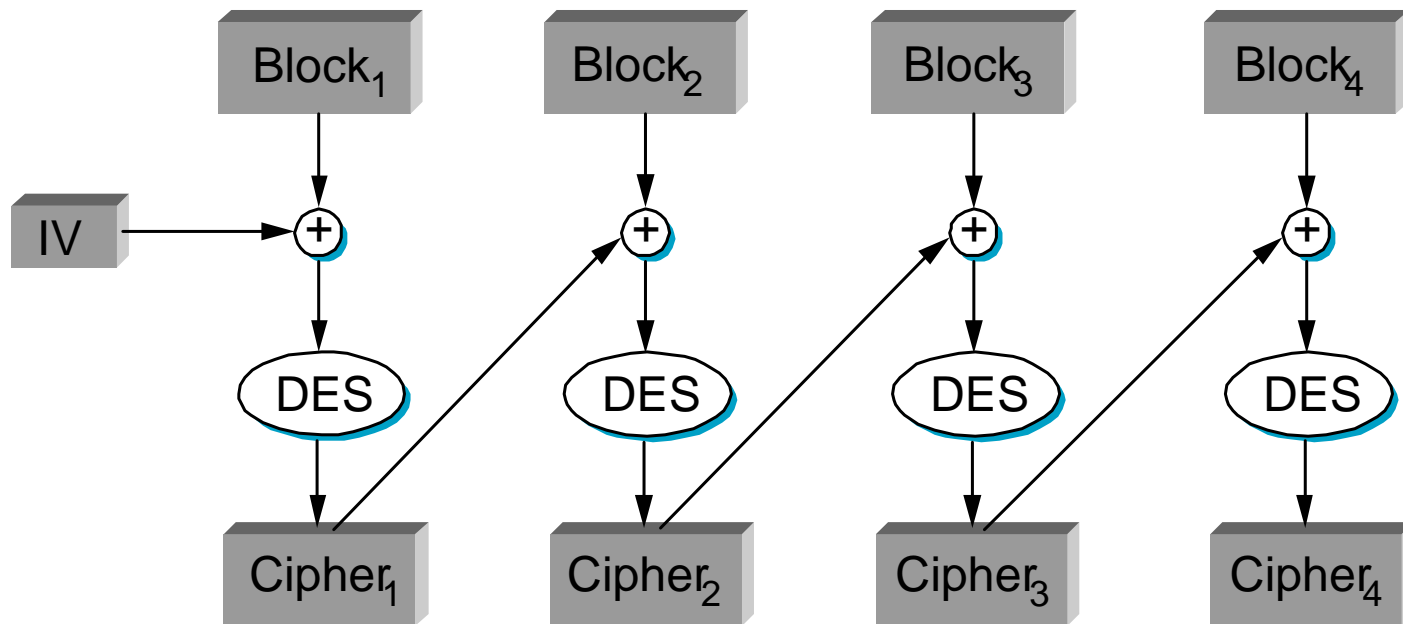
- 64-bit key (56-bits + 8-bit parity)
- 16 rounds



- Each Round



- Repeat for larger messages





# Public Key (RSA)



- Encryption & Decryption

$$c = m^e \bmod n$$

$$m = c^d \bmod n$$

# RSA (cont)

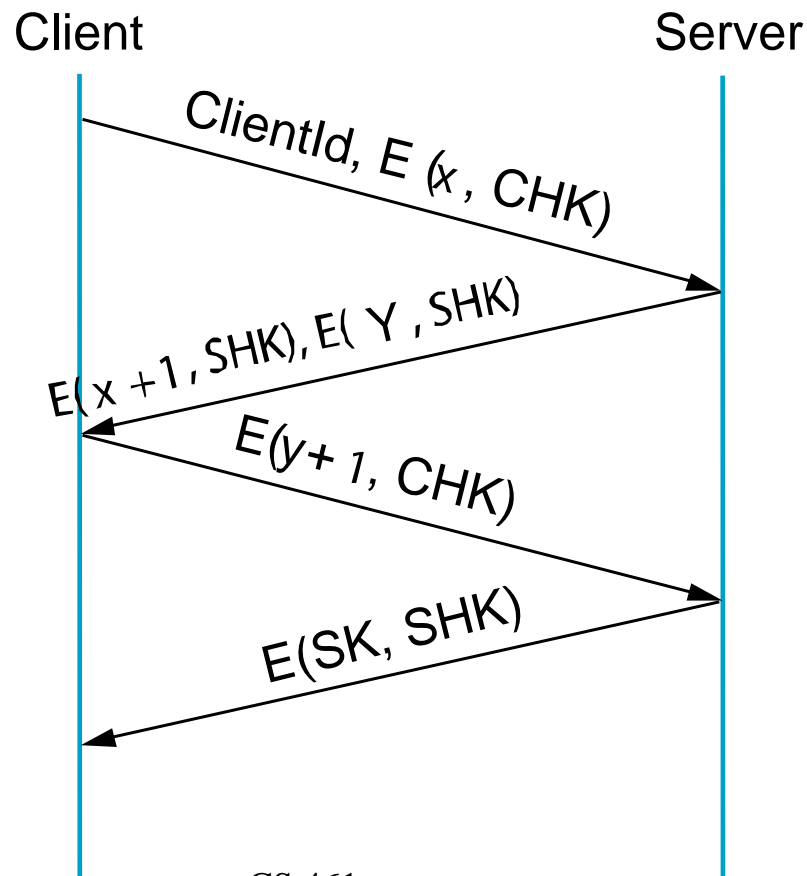
- Choose two large prime numbers  $p$  and  $q$  (each 256 bits)
- Multiply  $p$  and  $q$  together to get  $n$
- Choose the encryption key  $e$ , such that  $e$  and  $(p - 1) \times (q - 1)$  are relatively prime.
- Two numbers are relatively prime if they have no common factor greater than one
- Compute decryption key  $d$  such that
$$d = e^{-1} \bmod ((p - 1) \times (q - 1))$$
- Construct public key as  $(e, n)$
- Construct private key as  $(d, n)$
- Discard (do not disclose) original primes  $p$  and  $q$

# Message Digest

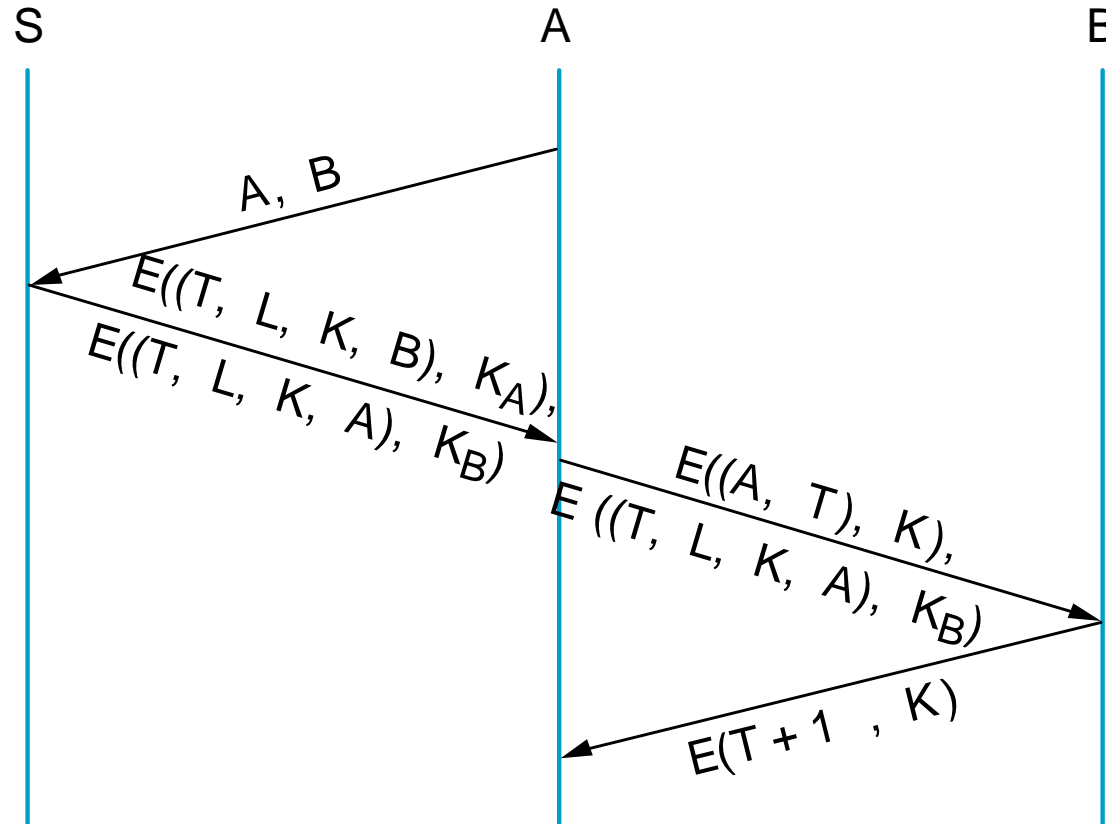
- Cryptographic checksum
  - just as a regular checksum protects the receiver from accidental changes to the message, a cryptographic checksum protects the receiver from malicious changes to the message.
- One-way function
  - given a cryptographic checksum for a message, it is virtually impossible to figure out what message produced that checksum; it is not computationally feasible to find two messages that hash to the same cryptographic checksum.
- Relevance
  - if you are given a checksum for a message and you are able to compute exactly the same checksum for that message, then it is highly likely this message produced the checksum you were given.

# Authentication Protocols

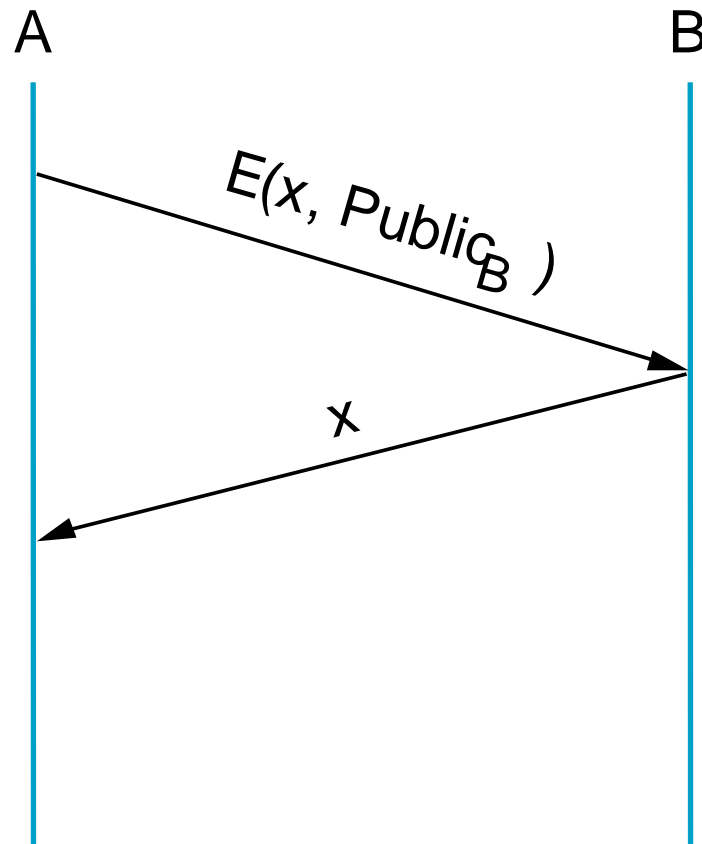
- Three-way handshake



- Trusted third party (Kerberos)



- Public key authentication



# Message Integrity Protocols

- Digital signature using RSA
  - special case of a message integrity where the code can only have been generated by one participant
  - compute signature with private key and verify with public key
- Keyed MD5
  - sender:  $m + \text{MD5}(m + k) + E(k, \text{private})$
  - receiver
    - recovers random key using the sender's public key
    - applies MD5 to the concatenation of this random key message
- MD5 with RSA signature
  - sender:  $m + E(\text{MD5}(m), \text{private})$
  - receiver
    - decrypts signature with sender's public key
    - compares result with MD5 checksum sent with message

# Message Integrity Protocols

- Digital signature using RSA
  - special case of a message integrity where the code can only have been generated by one participant
  - compute signature with private key and verify with public key
- Keyed MD5
  - sender:  $m + \text{MD5}(m + k) + \text{E}(\text{E}(k, \text{rcv-pub}), \text{private})$
  - receiver
    - recovers random key using the sender's public key
    - applies MD5 to the concatenation of this random key message
- MD5 with RSA signature
  - sender:  $m + \text{E}(\text{MD5}(m), \text{private})$
  - receiver
    - decrypts signature with sender's public key
    - compares result with MD5 checksum sent with message



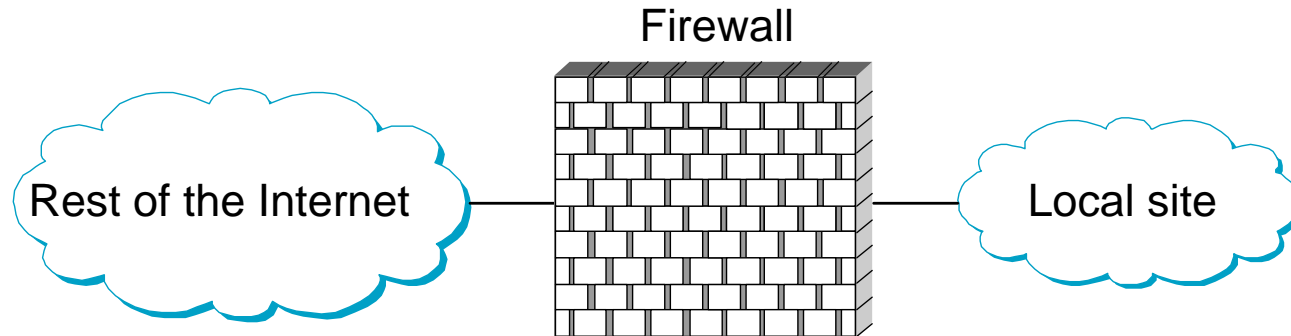
# Key Distribution

- Certificate
  - special type of digitally signed document:  
*“I certify that the public key in this document belongs to the entity named in this document, signed X.”*
  - the name of the entity being certified
  - the public key of the entity
  - the name of the certified authority
  - a digital signature
- Certified Authority (CA)
  - administrative entity that issues certificates
  - useful only to someone that already holds the CA’s public key.

# Key Distribution (cont)

- Chain of Trust
  - if  $X$  certifies that a certain public key belongs to  $Y$ , and  $Y$  certifies that another public key belongs to  $Z$ , then there exists a chain of certificates from  $X$  to  $Z$
  - someone that wants to verify  $Z$ 's public key has to know  $X$ 's public key and follow the chain
- Certificate Revocation List

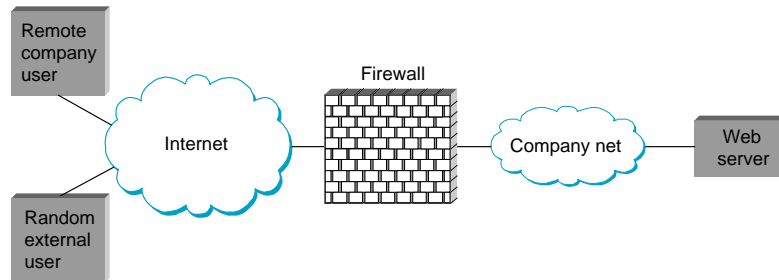
# Firewalls



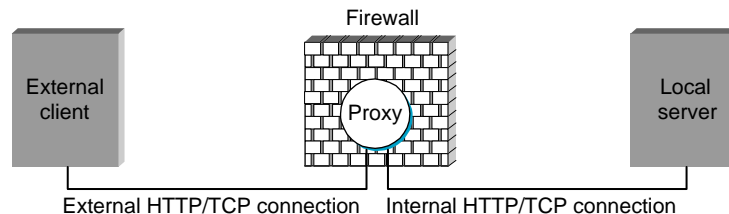
- Filter-Based Solution
  - example
    - ( 192.12.13.14, 1234, 128.7.6.5, 80 )
    - ( \*,\*, 128.7.6.5, 80 )
  - default: forward or not forward?
  - how dynamic?

# Proxy-Based Firewalls

- Problem: complex policy
- Example: web server



- Solution: proxy



- Design: transparent vs. classical
- Limitations: attacks from within

# Denial of Service

- Attacks on end hosts
  - SYN attack
- Attacks on routers
  - Christmas tree packets
  - pollute route cache
- Authentication attacks
- Distributed DoS attacks

# *Overlay Networks*

*Go To Talk Outline*

# Peer-to-Peer Networks

## Outline

- Survey

- Self-organizing overlay network

- File system on top of P2P network

## Contributions from Peter Druschel

# Background

- Distribution
- Decentralized control
- Self-organization
- Symmetric communication



# Examples

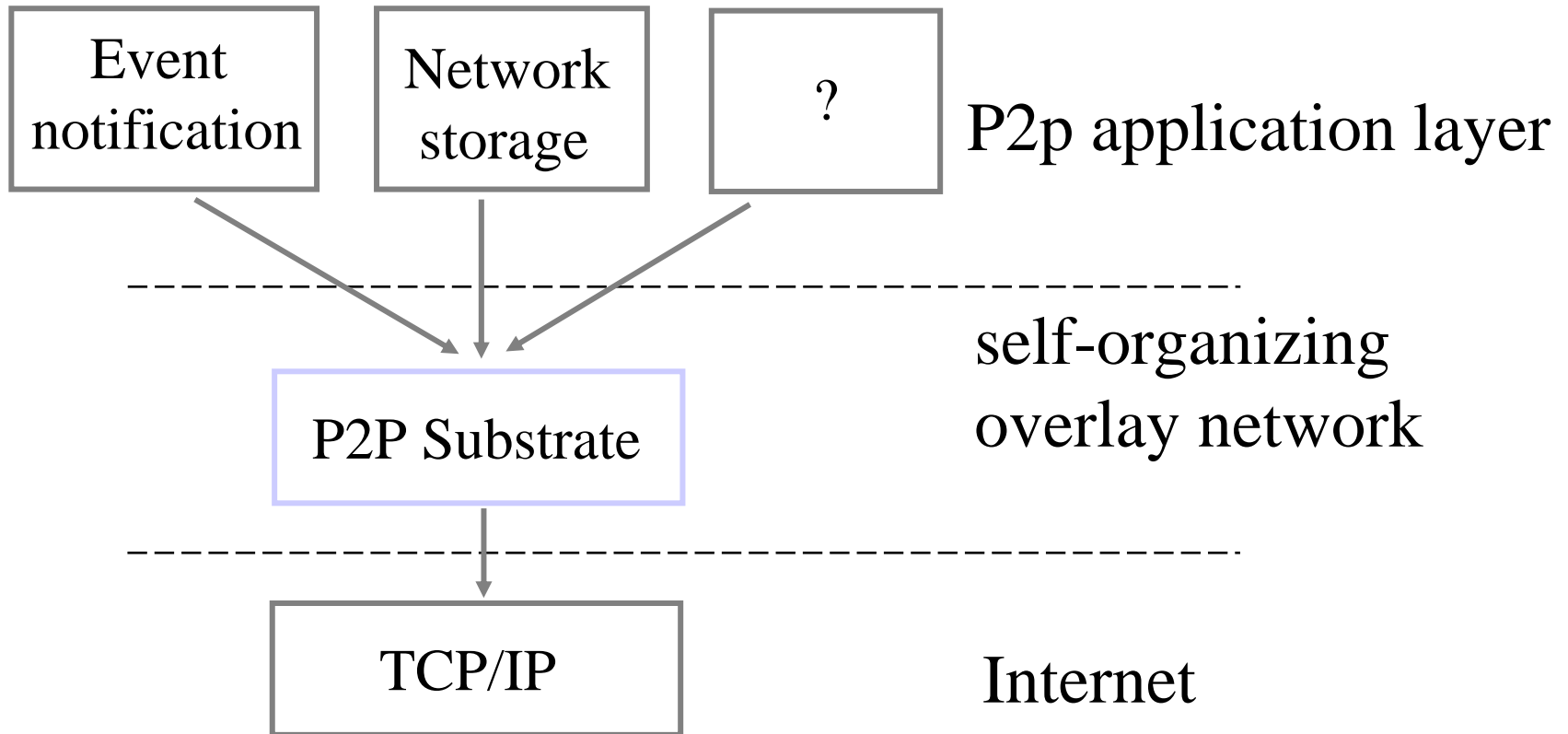
- Pioneers
  - Napster, Gnutella, FreeNet
- Academic Prototypes
  - Pastry, Chord, CAN,...

# Common Issues

- Organize, maintain overlay network
  - node arrivals
  - node failures
- Resource allocation/load balancing
- Resource location
- Locality (network proximity)

**Idea:** generic p2p substrate

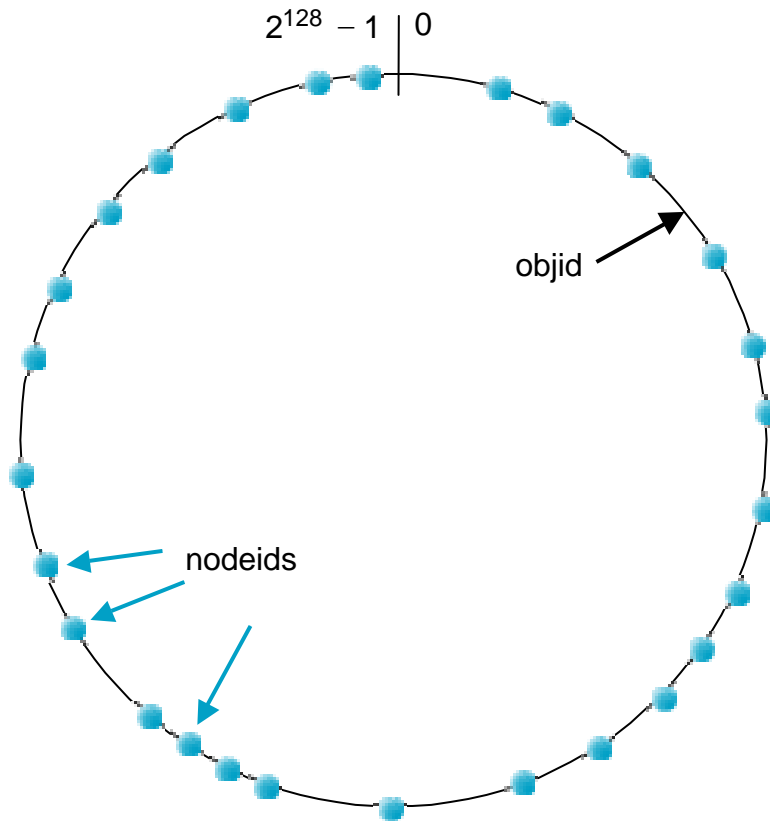
# Architecture



# Pastry

- Self-organizing overlay network
- Consistent hashing
- Lookup/insert object in  $< \log_{16} N$  routing steps (expected)
- $O(\log N)$  per-node state
- Network locality heuristics

# Object Distribution



## Consistent hashing [Karger et al. '97]

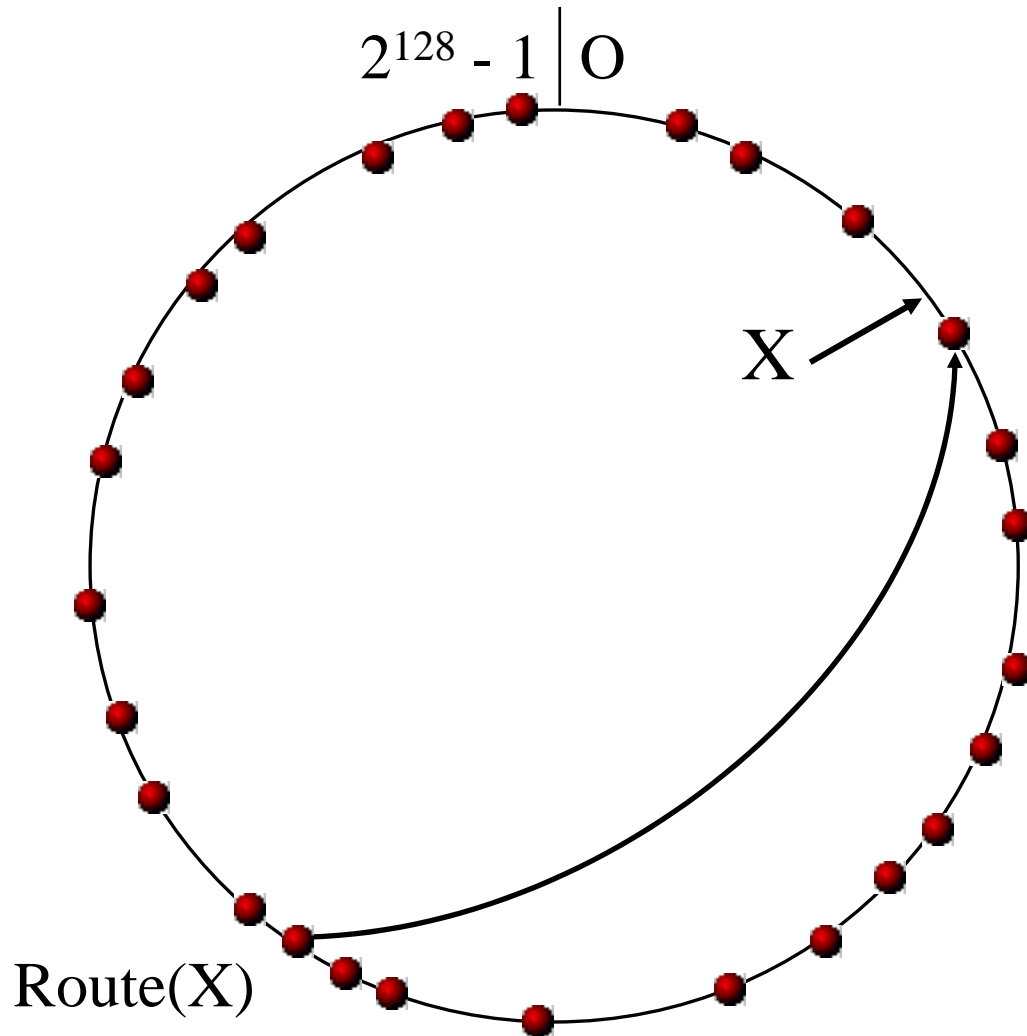
128 bit circular id space

*nodeIds* (uniform random)

*objIds* (uniform random)

**Invariant:** node with  
numerically closest nodeId  
maintains object

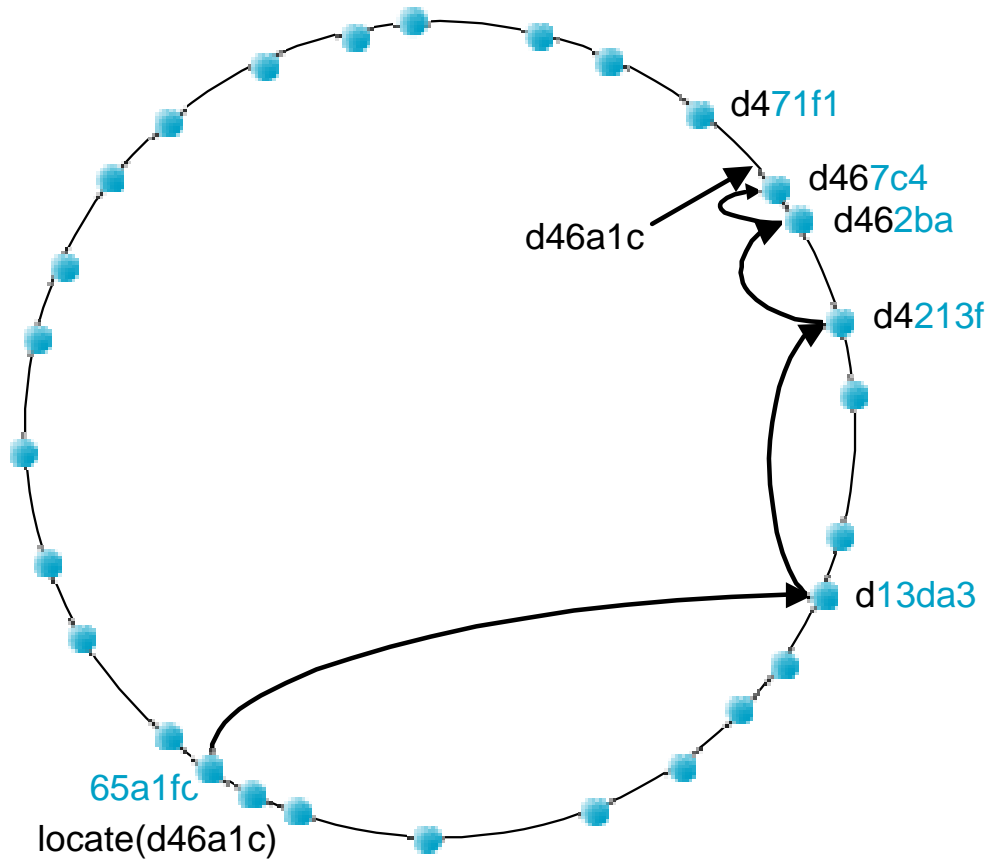
# Object Insertion/Lookup



Msg with key  $X$   
is routed to live  
node with nodeId  
closest to  $X$

**Problem:**  
complete routing  
table not feasible

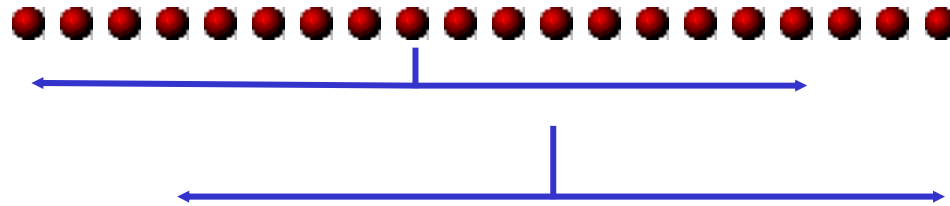
# Routing



## Properties

- $\log_{16} N$  steps
- $O(\log N)$  state

# Leaf Sets



*Each node maintains IP addresses of the nodes with the  $L$  numerically closest larger and smaller nodeIds, respectively.*

- routing efficiency/robustness
- fault detection (keep-alive)
- application-specific local coordination



# Routing Procedure

```
if (destination is within range of our leaf set)
    forward to numerically closest member
else
    let  $l$  = length of shared prefix
    let  $d$  = value of  $l$ -th digit in  $D$ 's address
    if ( $R_1^d$  exists)
        forward to  $R_1^d$ 
    else
        forward to a known node that
        (a) shares at least as long a prefix
        (b) is numerically closer than this node
```

# Routing

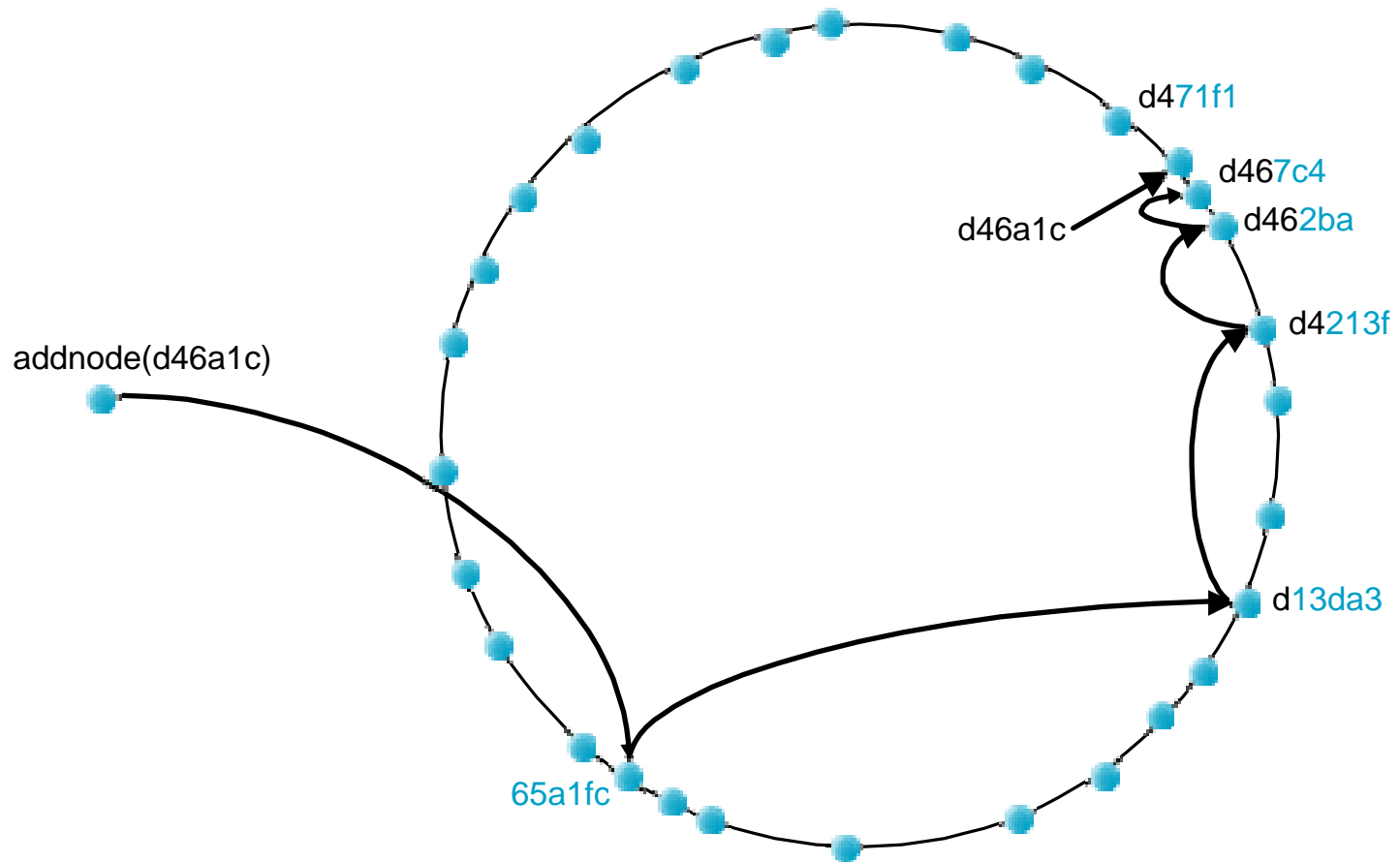
## **Integrity of overlay:**

- guaranteed unless  $L/2$  simultaneous failures of nodes with adjacent nodeIds

## **Number of routing hops:**

- No failures:  $< \log_{16} N$  expected,  $128/b + 1$  max
- During failure recovery:
  - $O(N)$  worst case, average case much better

# Node Addition



# Node Departure (Failure)

**Leaf set members exchange keep-alive messages**

- **Leaf set repair (eager):** request set from farthest live node in set
- **Routing table repair (lazy):** get table from peers in the same row, then higher rows

# API

- *route*( $M, X$ ): route message  $M$  to node with nodeId numerically closest to  $X$
- *deliver*( $M$ ): deliver message  $M$  to application
- *forwarding*( $M, X$ ): message  $M$  is being forwarded towards key  $X$
- *newLeaf*( $L$ ): report change in leaf set  $L$  to application

# PAST: Cooperative, archival file storage and distribution

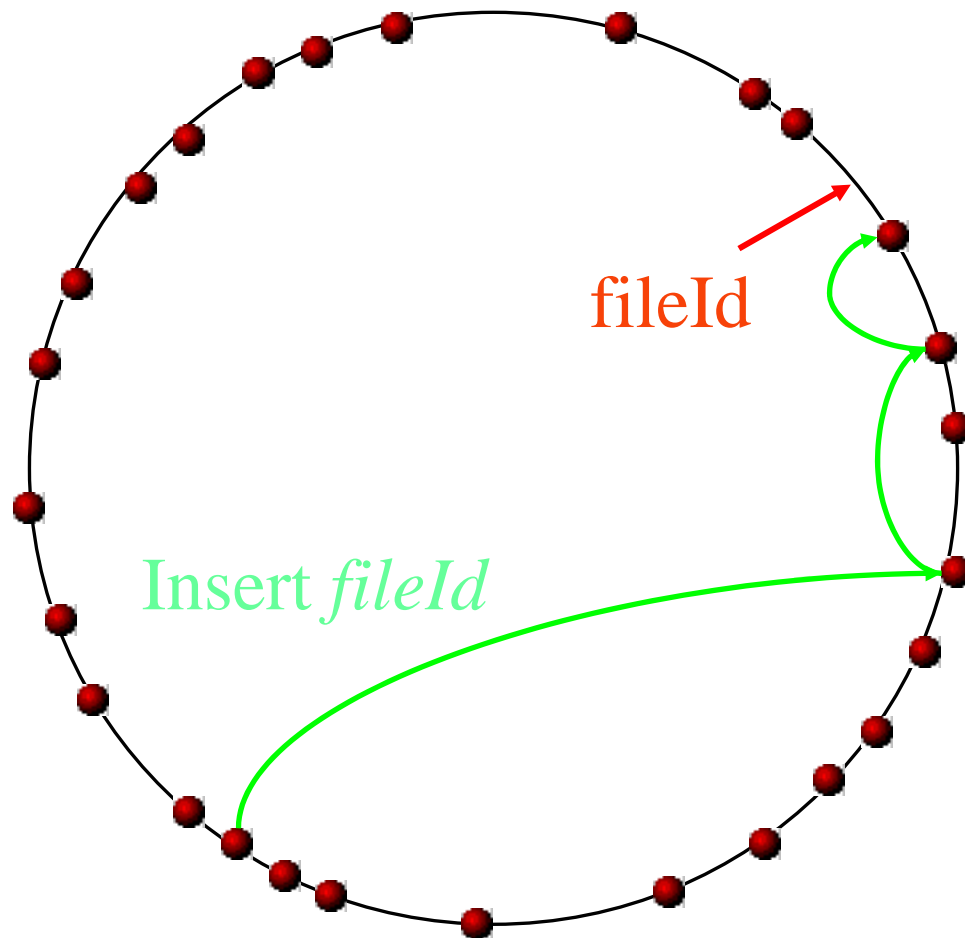
- Layered on top of Pastry
- Strong persistence
- High availability
- Scalability
- Reduced cost (no backup)
- Efficient use of pooled resources

# PAST API

- *Insert* - store replica of a file at  $k$  diverse storage nodes
- *Lookup* - retrieve file from a nearby live storage node that holds a copy
- *Reclaim* - free storage associated with a file

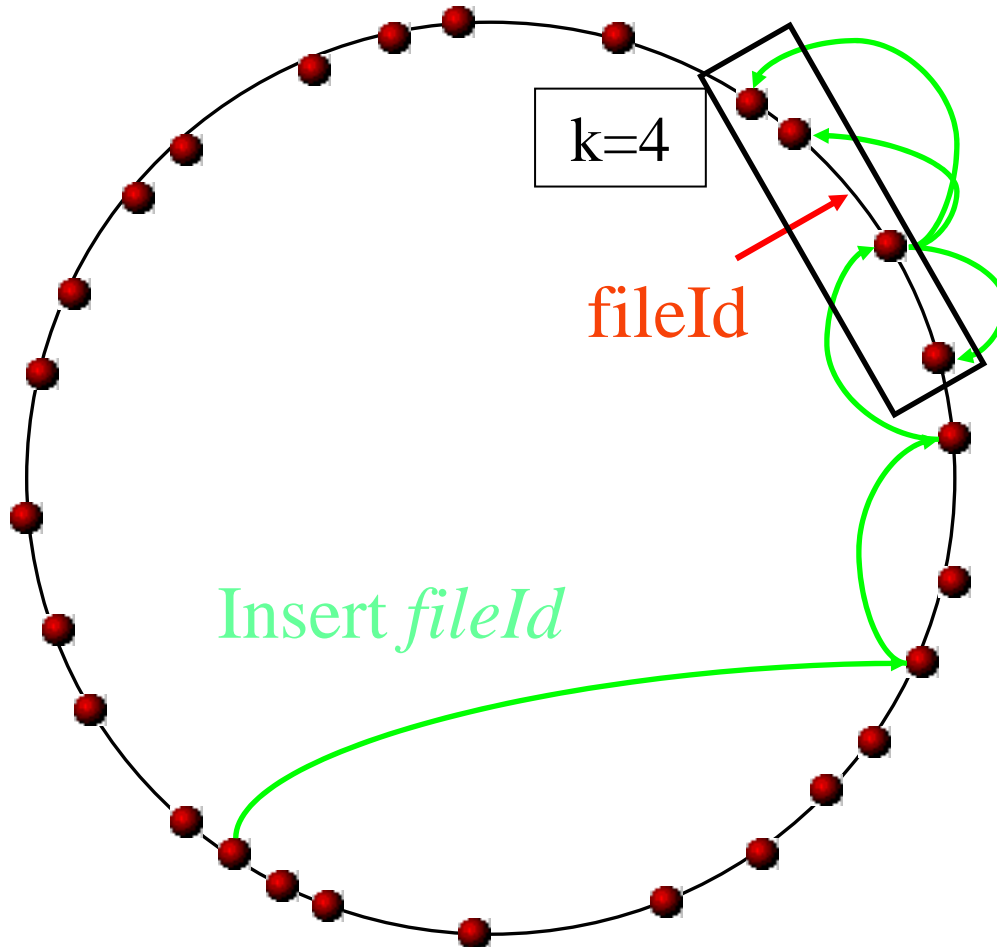
Files are *immutable*

# PAST: File storage





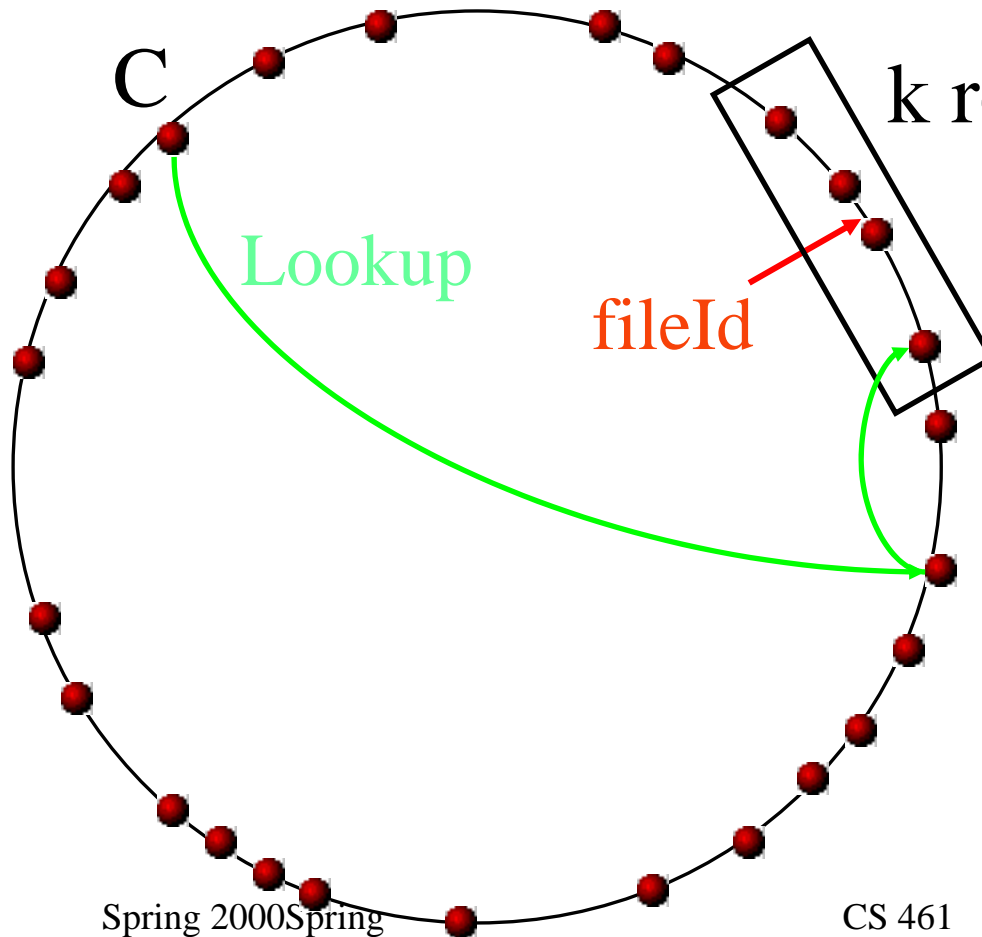
# PAST: File storage



**Storage Invariant:**  
File “replicas” are  
stored on  $k$  nodes  
with nodeIds  
closest to *fileId*

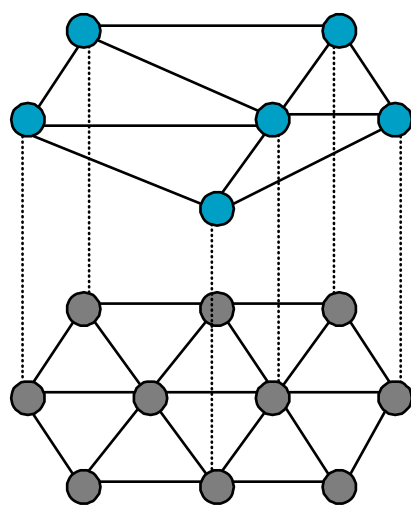
( $k$  is bounded by the  
leaf set size)

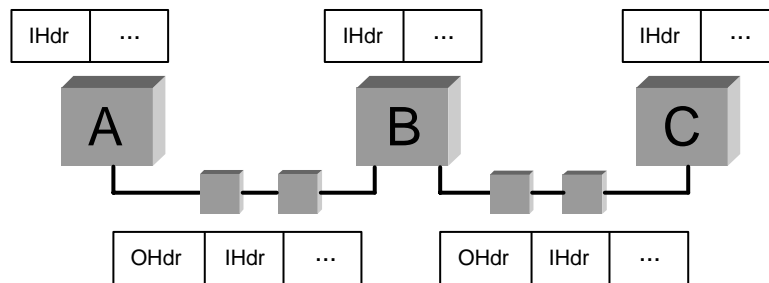
# PAST: File Retrieval

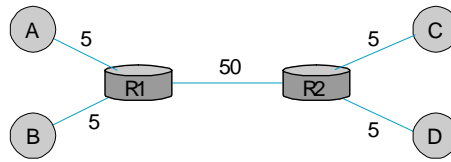


file located in  $\log_{16} N$  steps (expected)

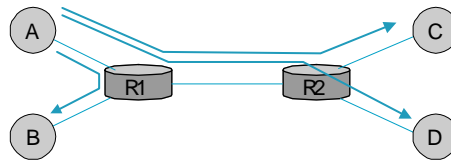
usually locates replica nearest client C



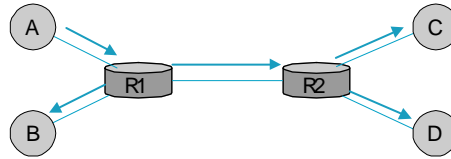




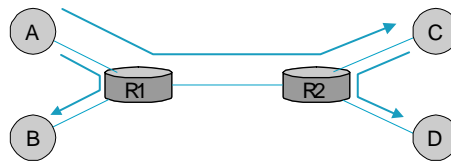
(a)



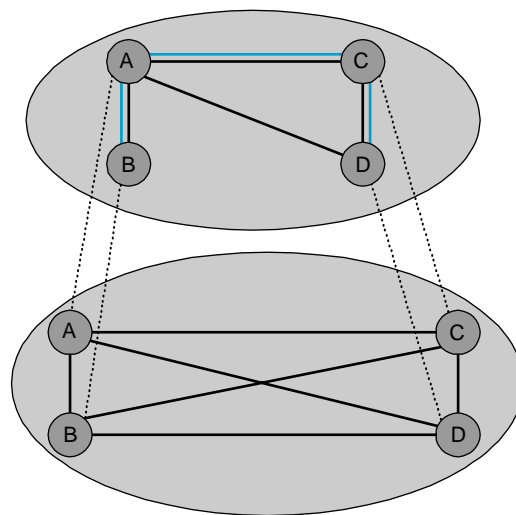
(b)

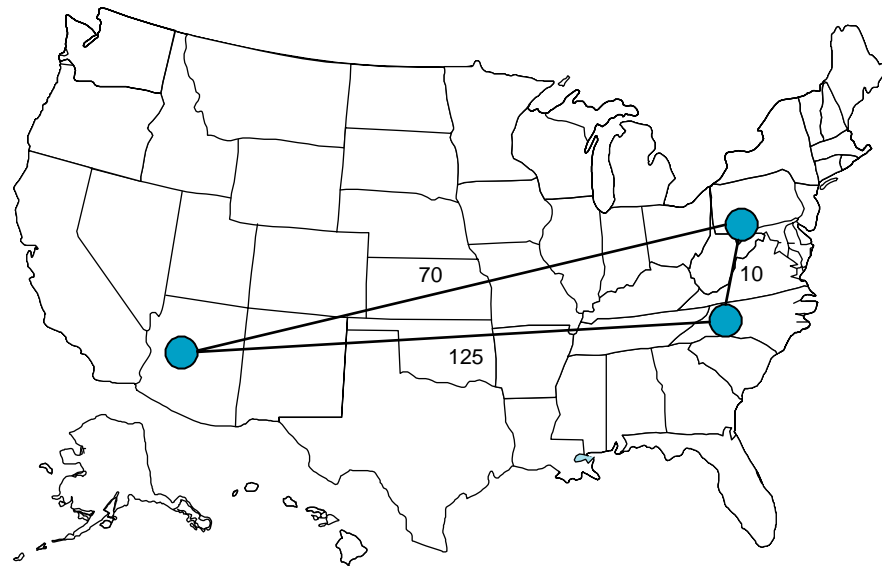


(c)



(d)





# Content Distribution Networks

## Outline

Implementation Techniques

Hashing Schemes

Redirection Strategies



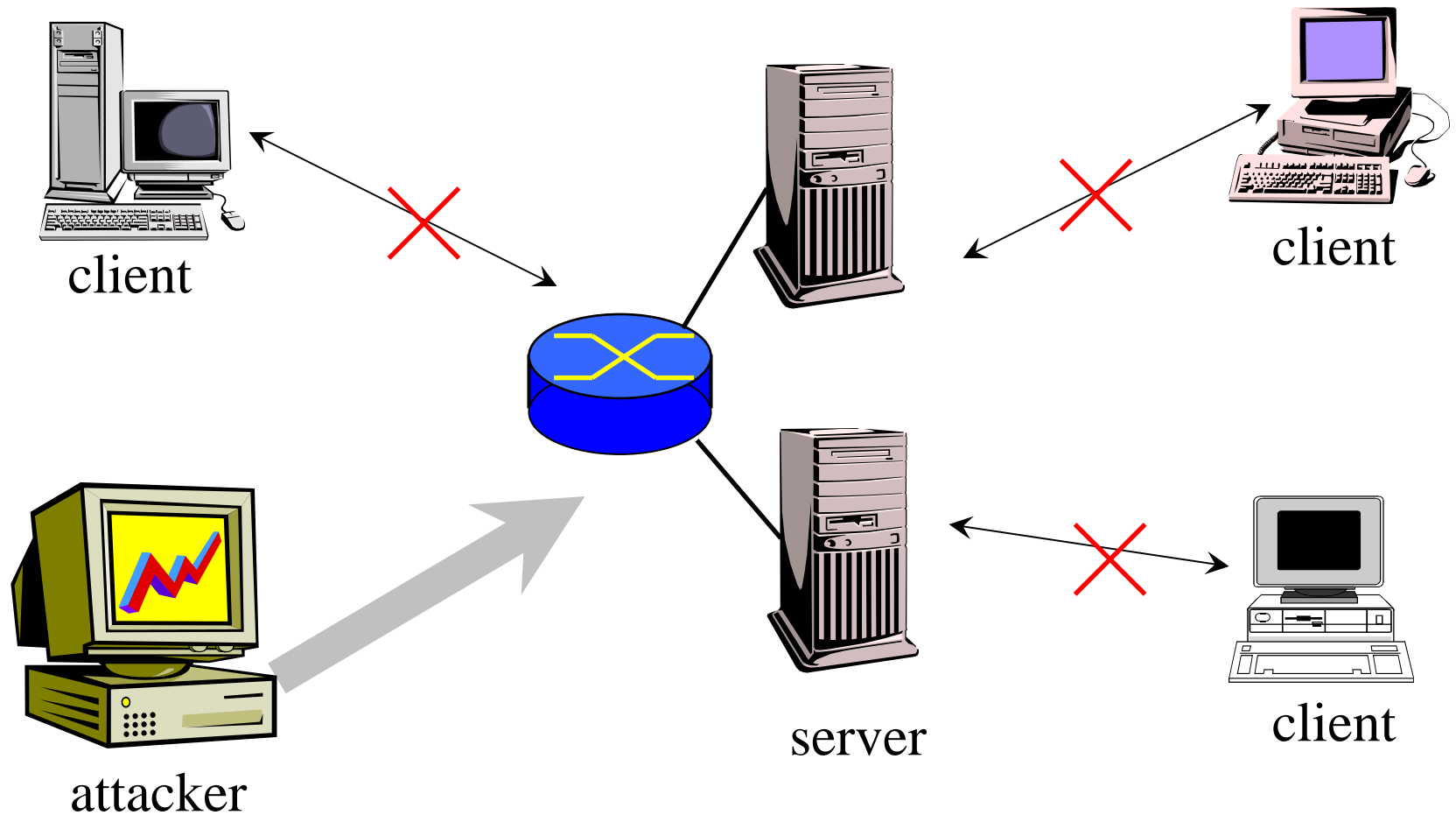
# Design Space

- Caching
  - explicit
  - transparent (hijacking connections)
- Replication
  - server farms
  - geographically dispersed (CDN)

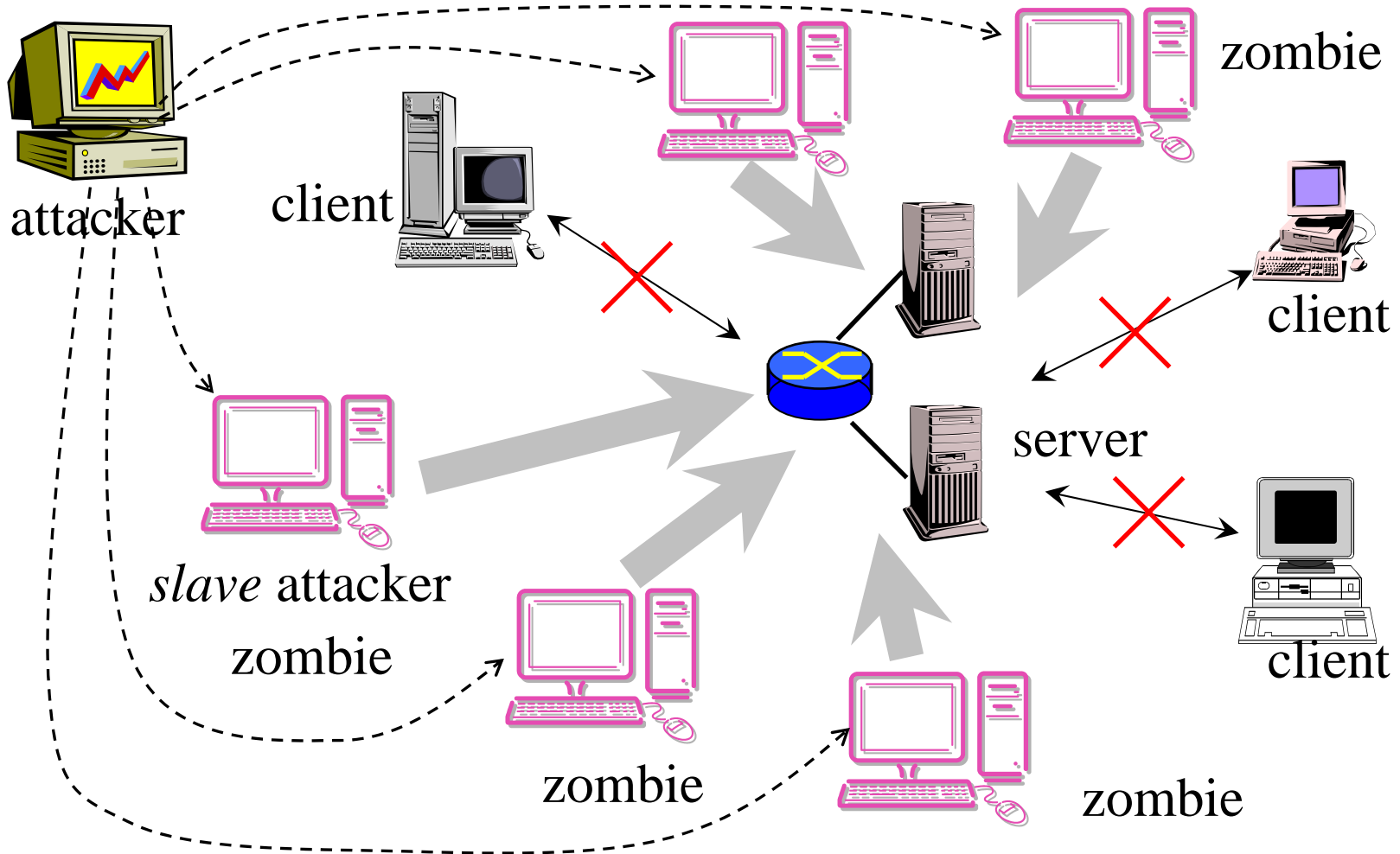
# Story for CDNs

- Traditional: *Performance*
  - move content closer to the clients
  - avoid server bottlenecks
- New: *DDoS Protection*
  - dissipate attack over massive resources
  - multiplicatively raise level of resources needed to attack

# Denial of Service Attacks (DoS)

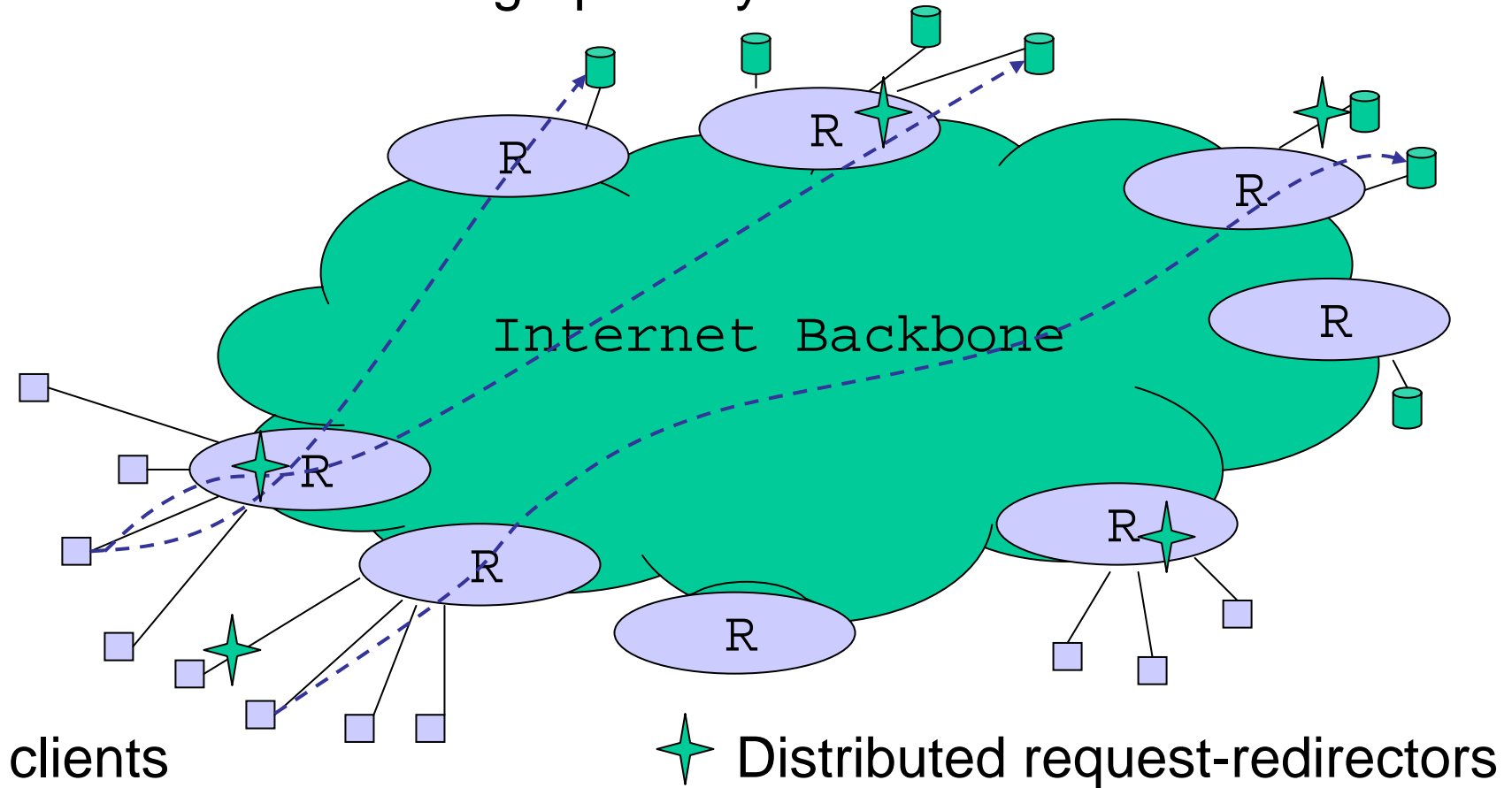


# Distributed DoS (DDoS)



# Redirection Overlay

Geographically distributed server clusters



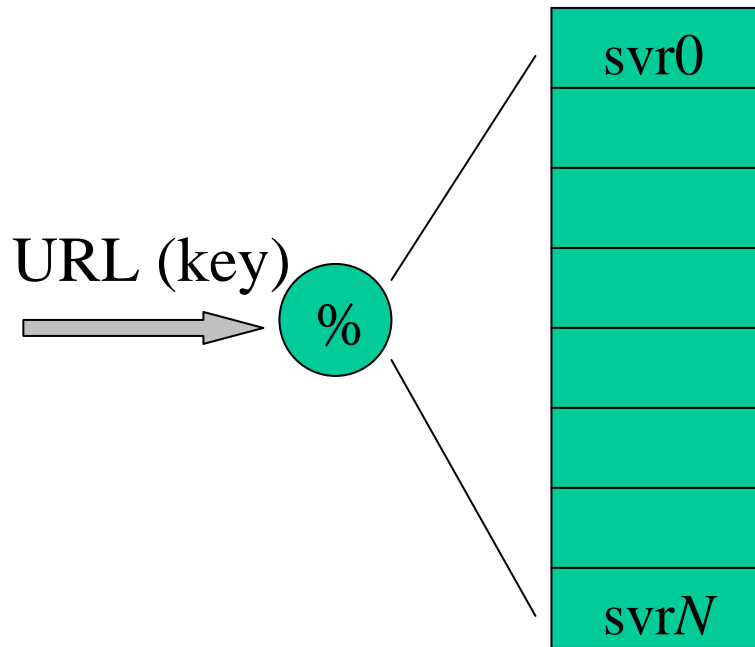
# Techniques

- DNS
  - one name maps onto many addresses
  - works for both servers and reverse proxies
- HTTP
  - requires an extra round trip
- Router
  - one address, select a server (reverse proxy)
  - content-based routing (near client)
- URL Rewriting
  - embedded links

# Redirection: Which Replica?

- Balance Load
- Cache Locality
- Network Delay

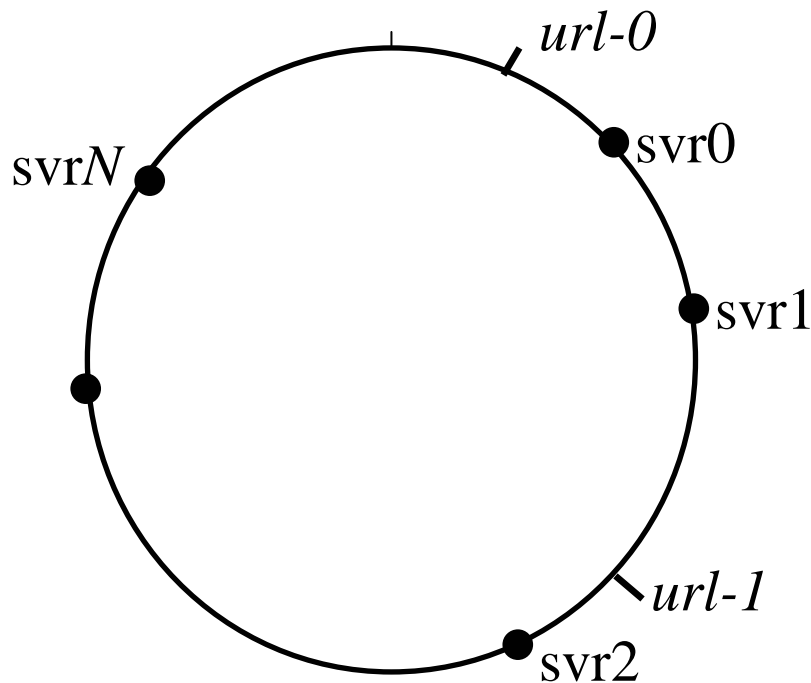
# Hashing Schemes: Modulo



- Easy to compute
- Evenly distributed
- Good for fixed number of servers
- Many mapping changes after a single server change



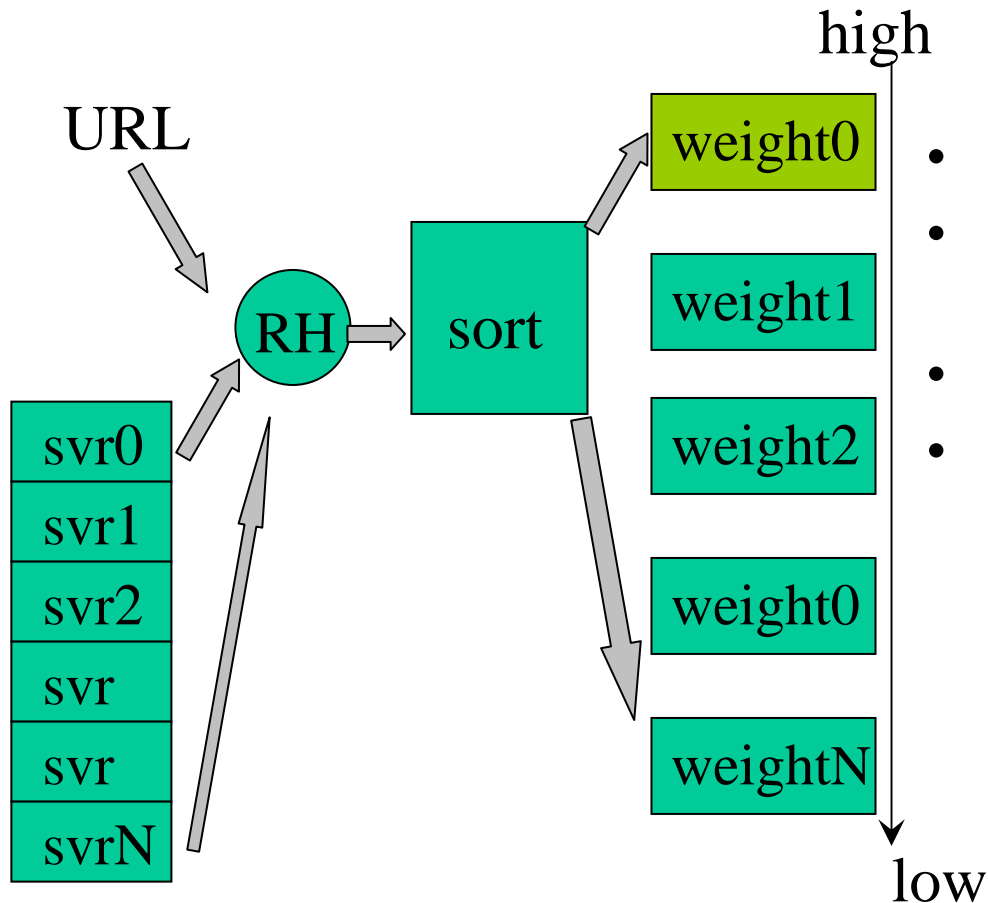
# Consistent Hashing (CHash)



Unit circle

- Hash server, then URL
- Closest match
- Only local mapping changes after adding or removing servers
- Used by State-of-the-art CDNs

# Highest Random Weight (HRW)



- Hash(url, svrAddr)
- Deterministic order of access set of servers
- Different order for different URLs
- Load evenly distributed after server changes

# Redirection Strategies

- Random (Rand)
  - Requests randomly sent to cooperating servers
  - Baseline case, no pathological behavior
- Replicated Consistent Hashing (R-CHash)
  - Each URL hashed to a fixed # of server replicas
  - For each request, randomly select one replica
- Replicated Highest Random Weight (R-HRW)
  - Similar to R-CHash, but use HRW hashing
  - Less likely two URLs have same set of replicas

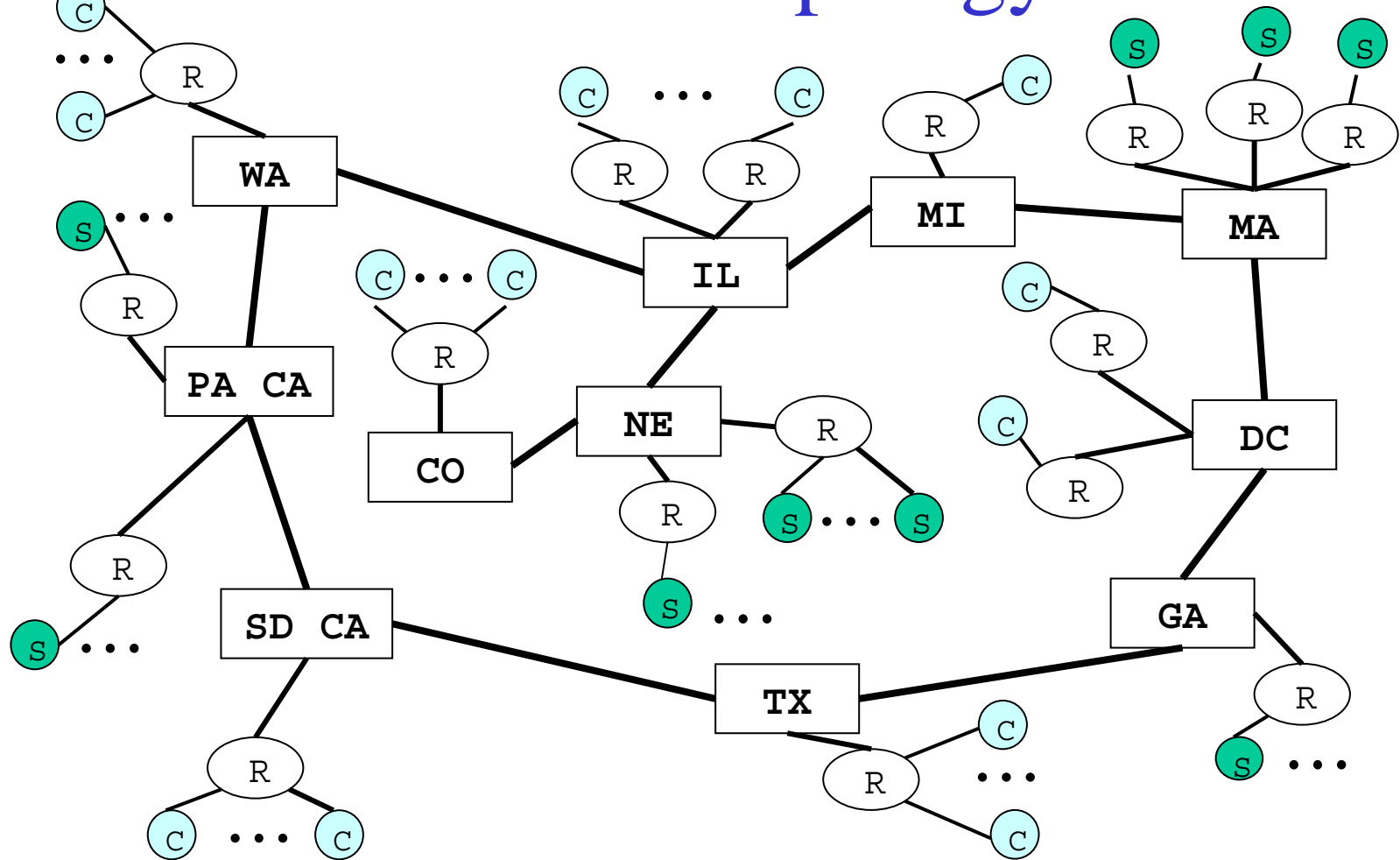
# Redirection Strategies (cont)

- Coarse Dynamic Replication (CDR)
  - Using HRW hashing to generate ordered server list
  - Walk through server list to find a lightly loaded one
  - # of replicas for each URL dynamically adjusted
  - Coarse grained server load information
- Fine Dynamic Replication (FDR)
  - Bookkeeping min # of replicas of URL (popularity)
  - Let more popular URL use more replicas
  - Keep less popular URL from extra replication

# Simulation

- Identifying bottlenecks
  - Server overload, network congestion...
- End-to-end network simulator prototype
  - Models network, application, and OS
  - Built on NS + LARD simulators
  - 100s of servers, 1000s of clients
  - >60,000 req/s using full-TCP transport
  - Measure capacity, latency, and scalability

# Network Topology



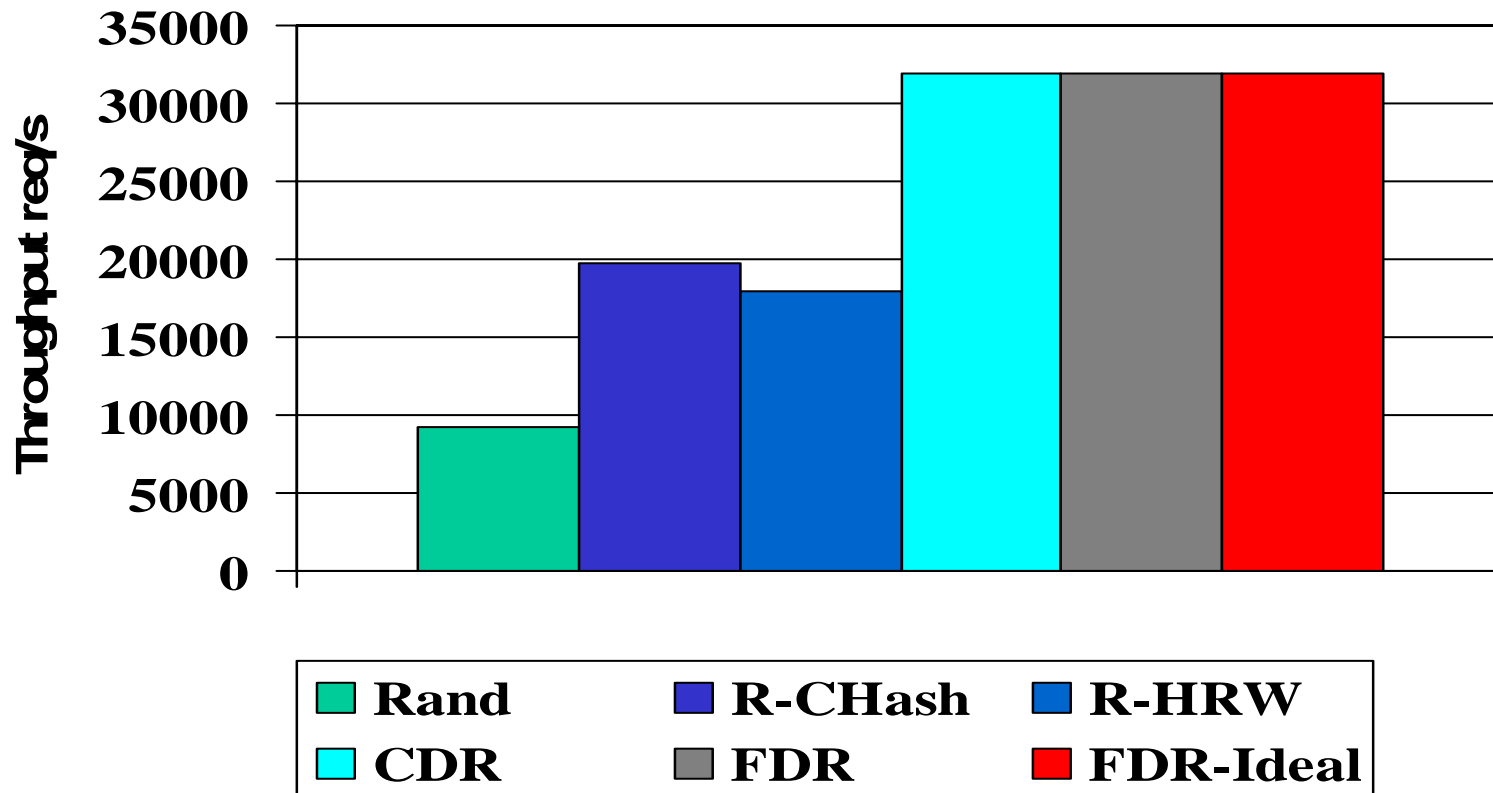
S – Server, C – Client, R - Router

# Simulation Setup

- Workload
  - Static documents from Web Server trace, available at each cooperative server
  - Attackers from random places, repeat requesting a subset of random files
- Simulation process
  - Gradually increase offered request load
  - End when servers very heavily overloaded

# Capacity: 64 server case

## Normal Operation

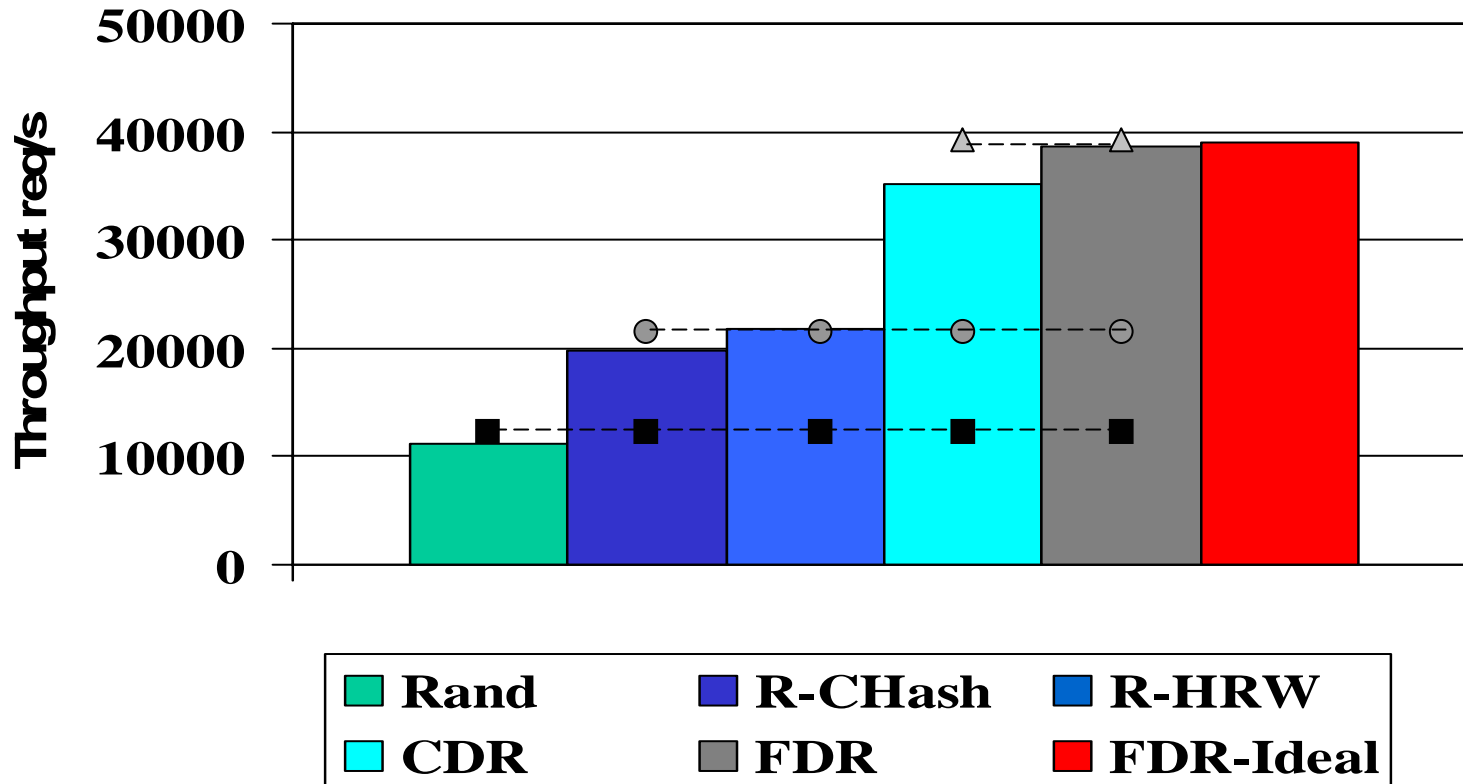


A single server can handle ~600 req/s in simulation



# Capacity: 64 server case

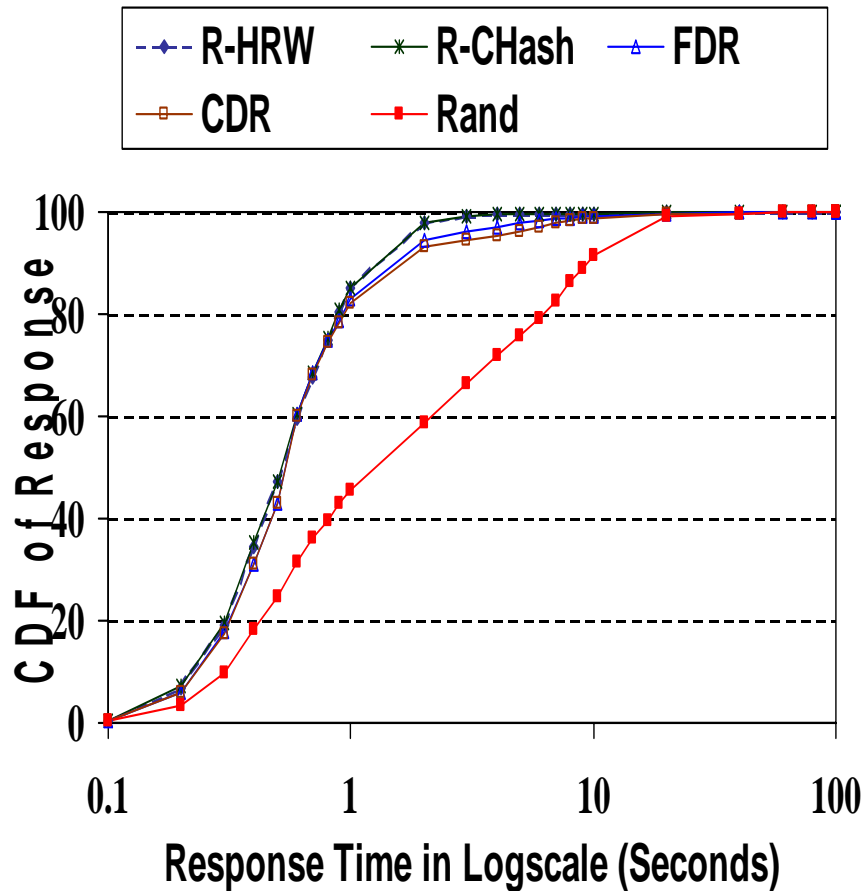
Under Attack (250 zombies, 10 files, avg 6KB)



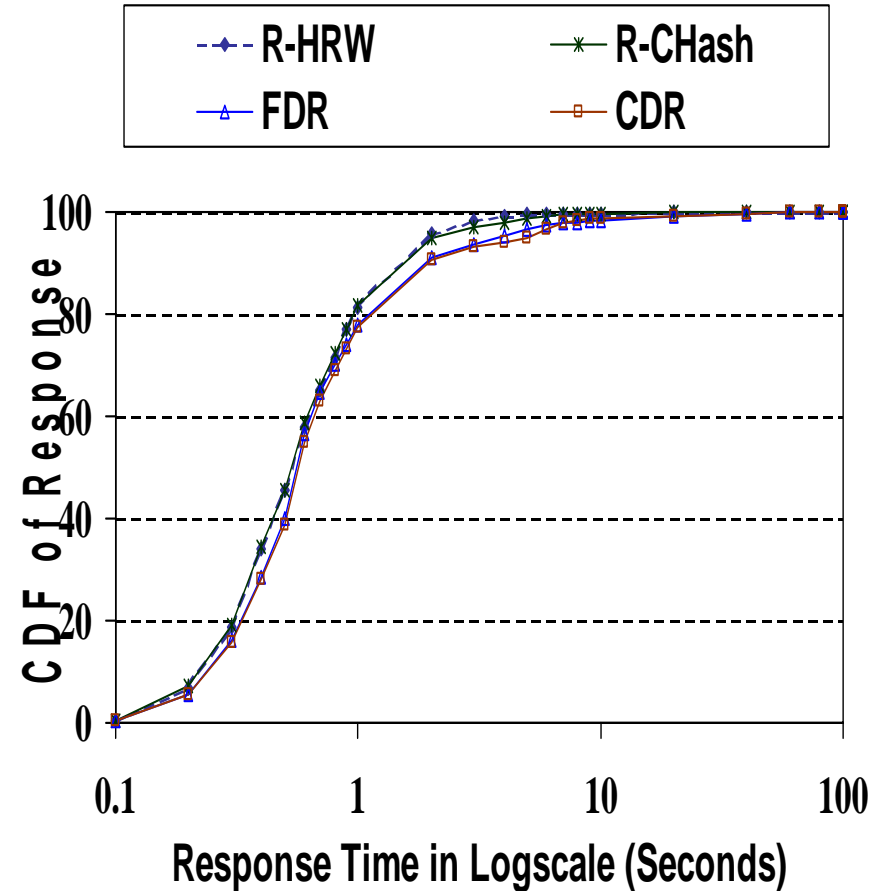
A single server can handle ~600 req/s in simulation

# Latency: 64 Servers Under Attack

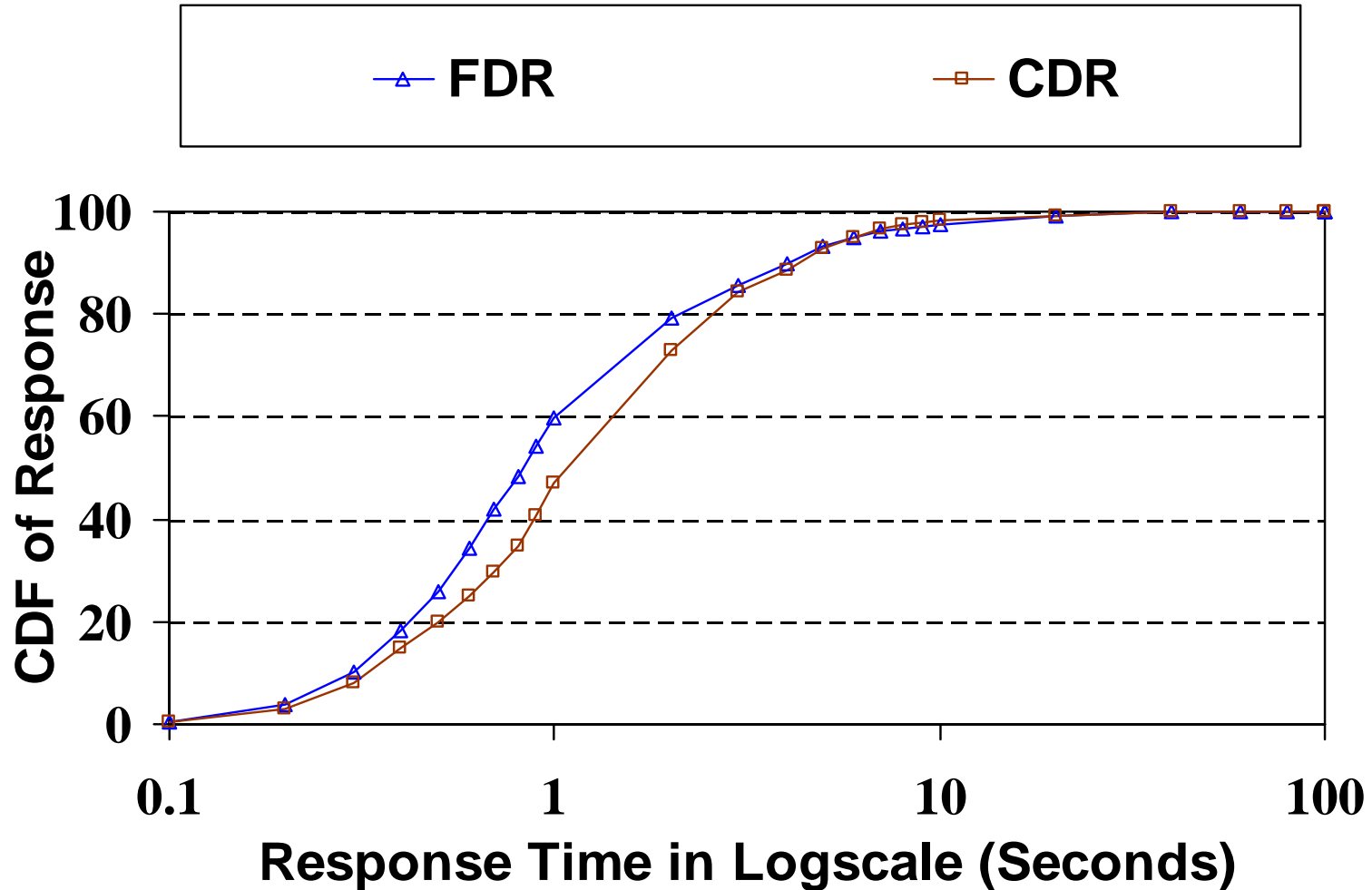
Random's Max: 11.2k req/s



R-CHash Max: 19.8k req/s



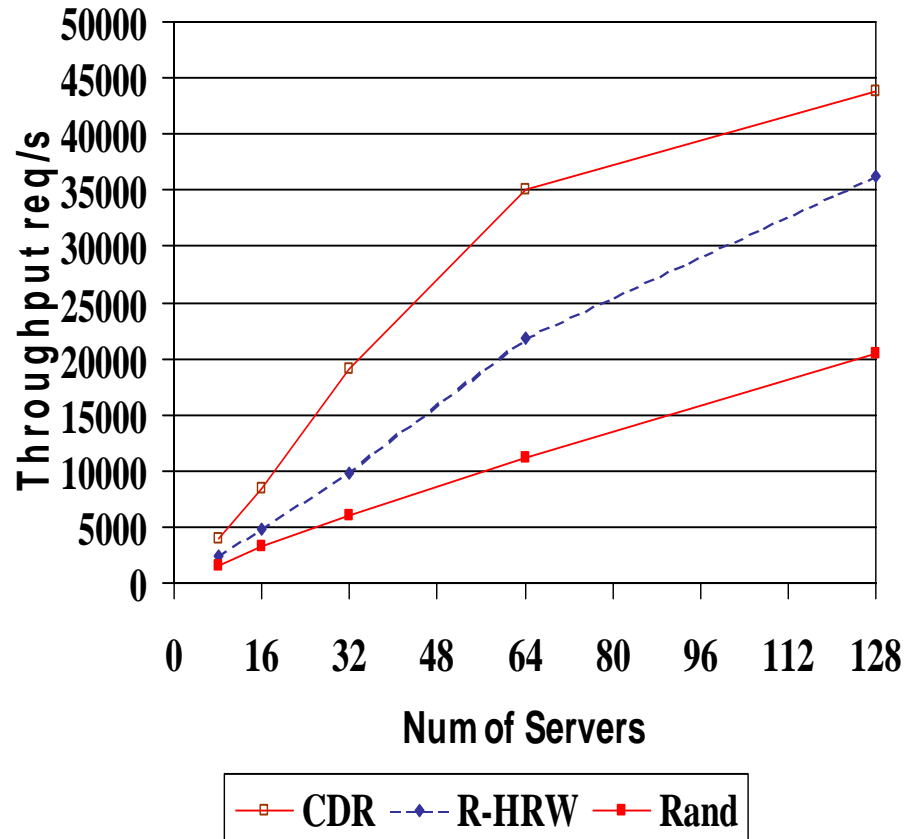
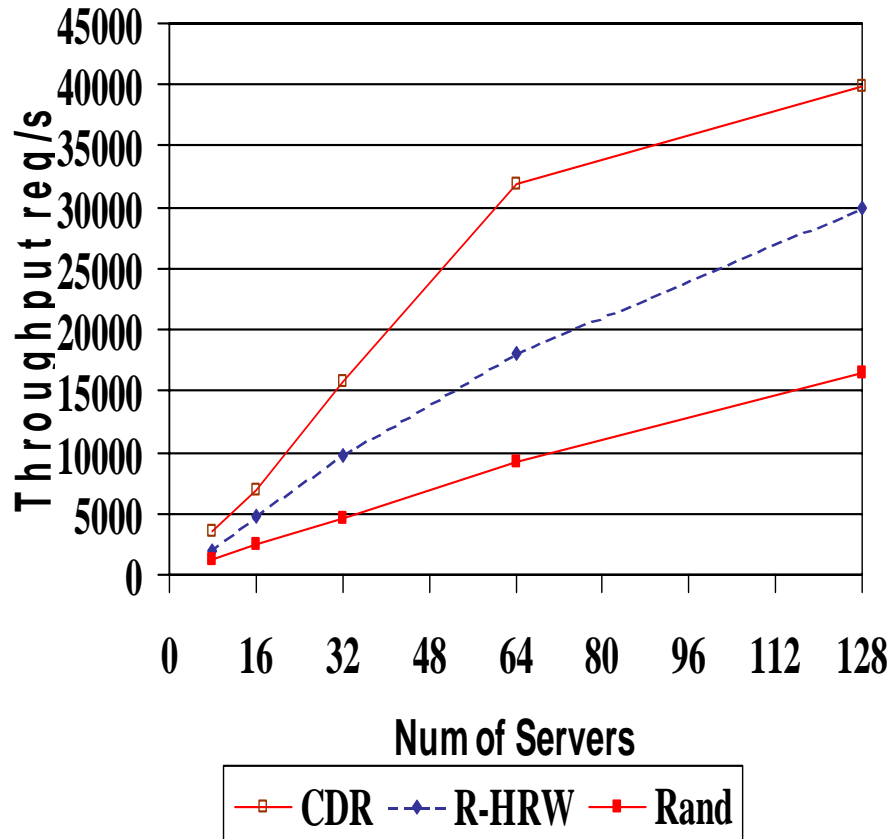
# Latency At CDR's Max: 35.1k req/s



# Capacity Scalability

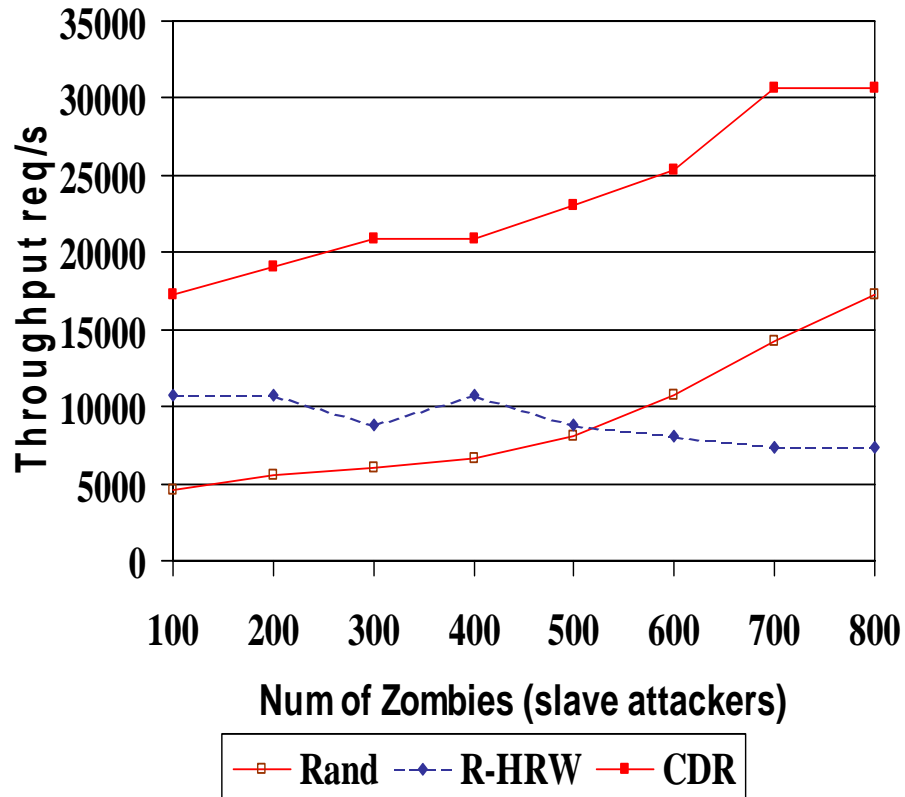
Normal Operation

Under Attack (250 zombies, 10 files)

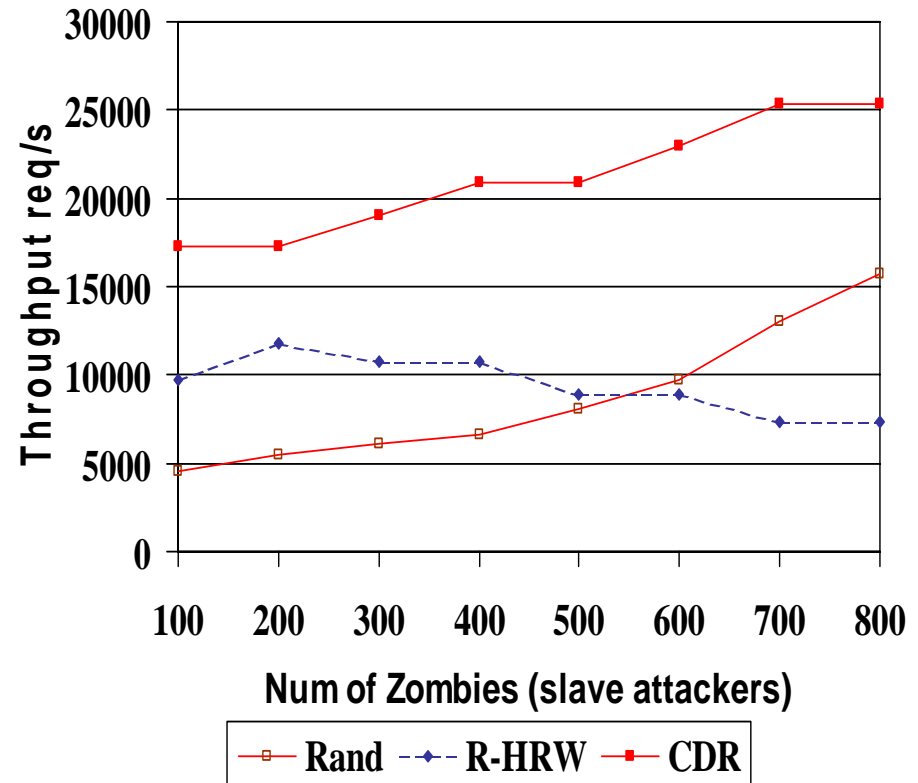


# Various Attacks (32 servers)

1 victim file, 1 KB



10 victim files, avg 6KB



# Deployment Issues

- Servers join DDoS protection overlay
  - Same story as Akamai
  - Get protection and performance
- Clients use DDoS protection service
  - Same story as proxy caching
  - Incrementally deployable
  - Get faster response and help others