

画像情報特論 (3)

- TCP/IP (2)

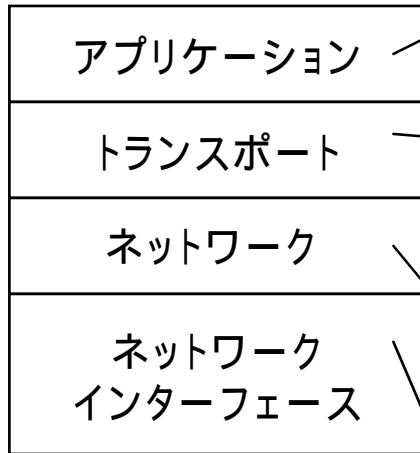
- TCP (Transport Control Protocol)
- UDP (User Datagram Protocol)
- DCCP (Datagram Congestion Control Protocol)

情報ネットワーク専攻 甲藤二郎

E-Mail: katto@waseda.jp

インターネットの基礎

プロトコルスタック



アプリケーション

HTTP, RTSP, FTP, Telnet, ...

← **RTP**: 実時間メディア用途

端末・端末間

TCP: 誤り訂正、順序制御、フロー制御 ... 信頼性重視

UDP: オーバーヘッド少 ... 低遅延、高速性重視

DCCP: TCP フレンドリ

端末・ルータ間、ルータ・ルータ間

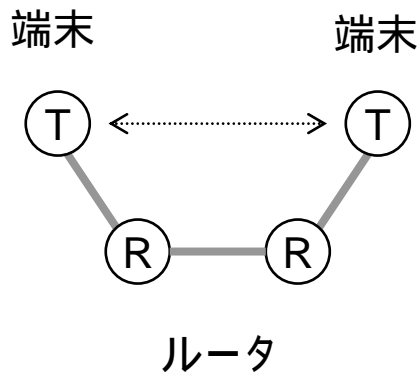
IP: 経路制御、フラグメンテーション

ICMP: エラー通知

IGMP: マルチキャスト (mbone)

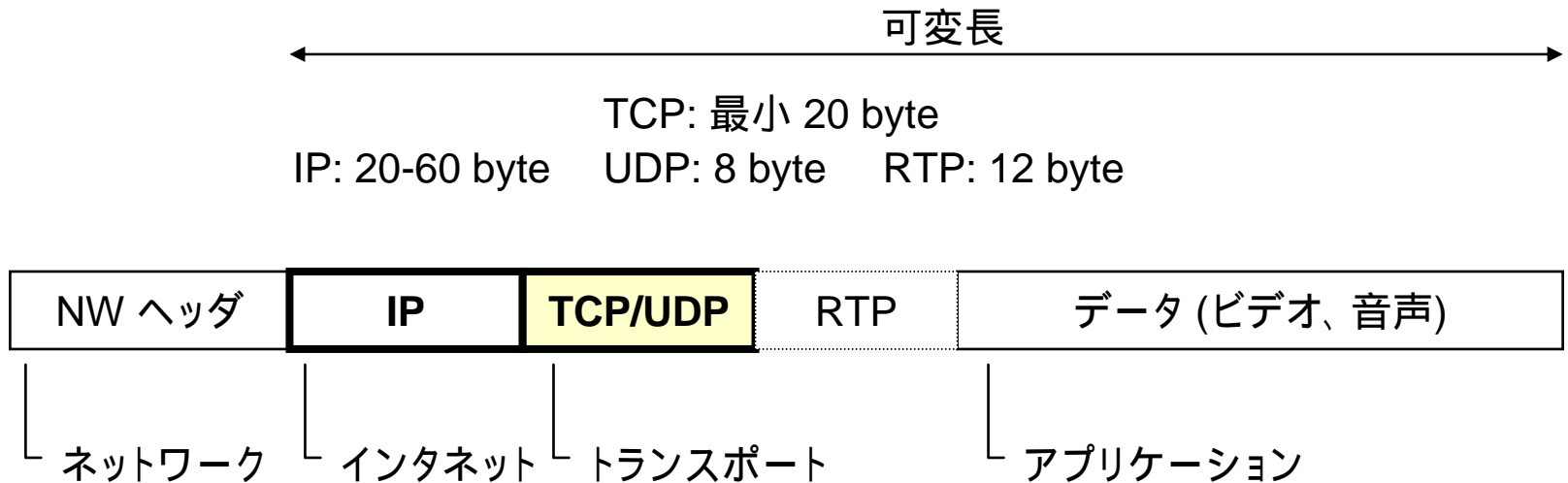
個別リンク

イーサネット, PPP, X.25, ATM, ...



IP データグラム

IP データグラム

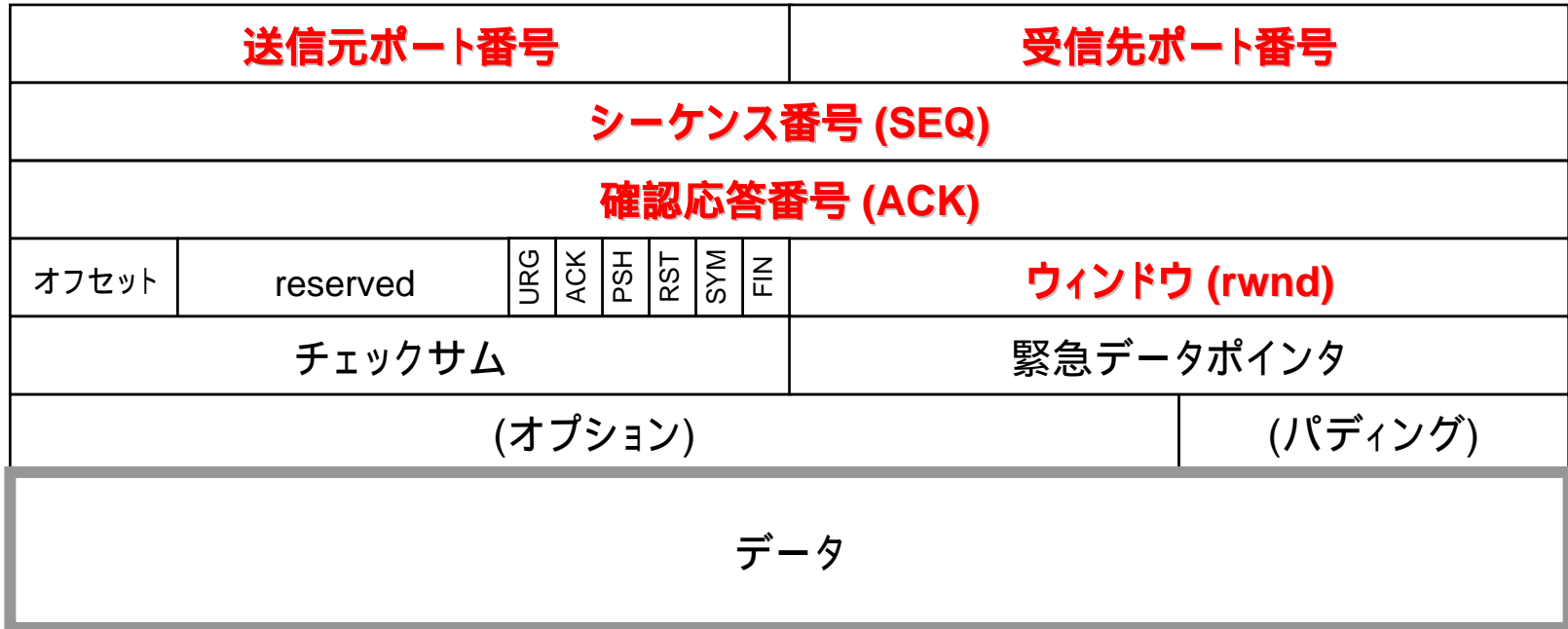


TCP

Transport Control Protocol

TCP ヘッダ

4 byte



ポート番号: アプリケーションの識別

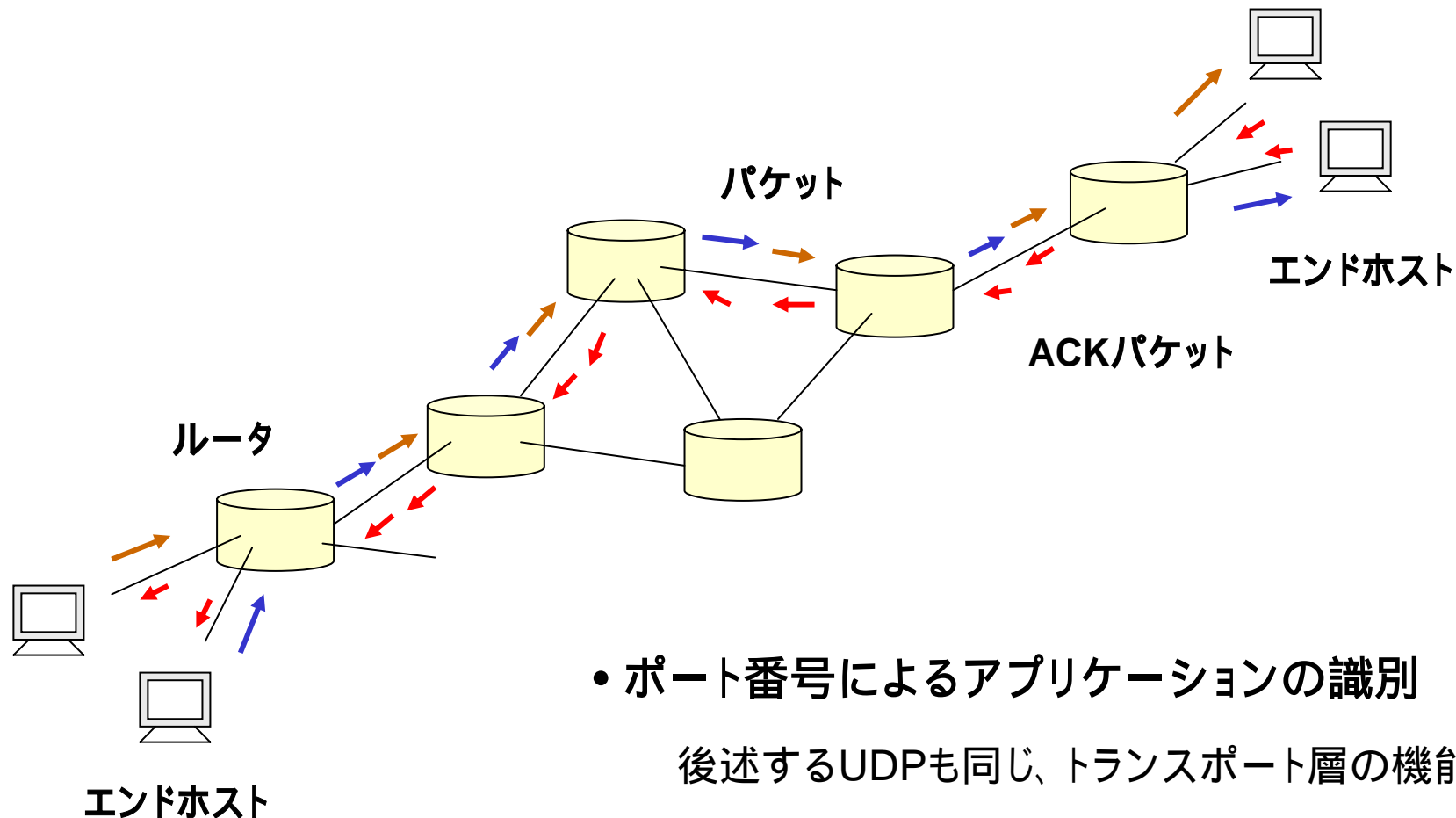
シーケンス番号: パケット廃棄、順序逆転を検出 (バイト単位でカウント)

確認応答番号: 次パケットで受信予定のシーケンス番号、あるいは重複 ACK の通知

ウィンドウ: 受信者が求める最大セグメントサイズ

TCP の機能

- End-to-End の確認応答による誤り制御とフロー制御



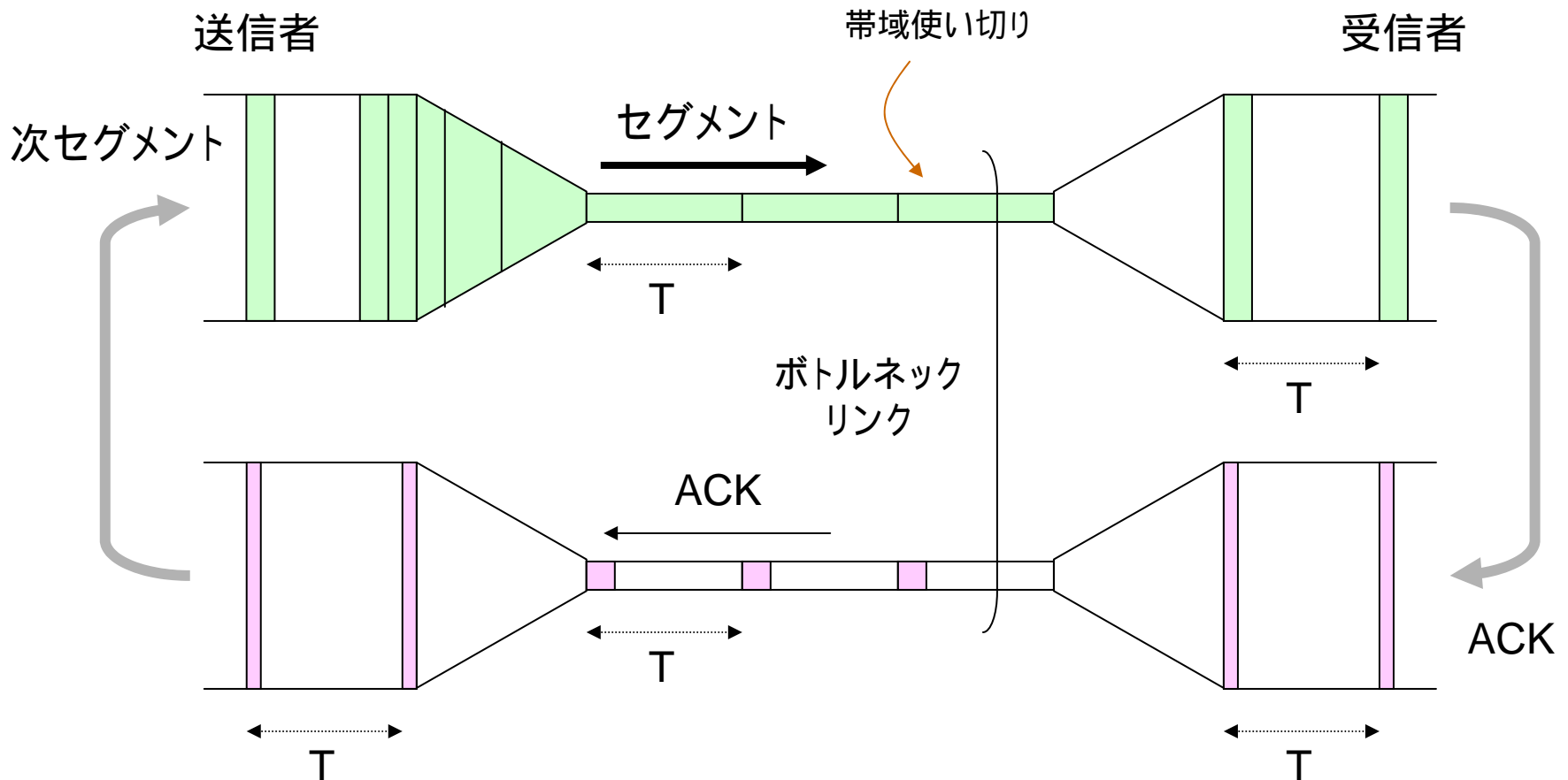
- ポート番号によるアプリケーションの識別

後述するUDPも同じ、トランスポート層の機能
いわゆる well-known port など

セルフ・クロッキング

self clocking

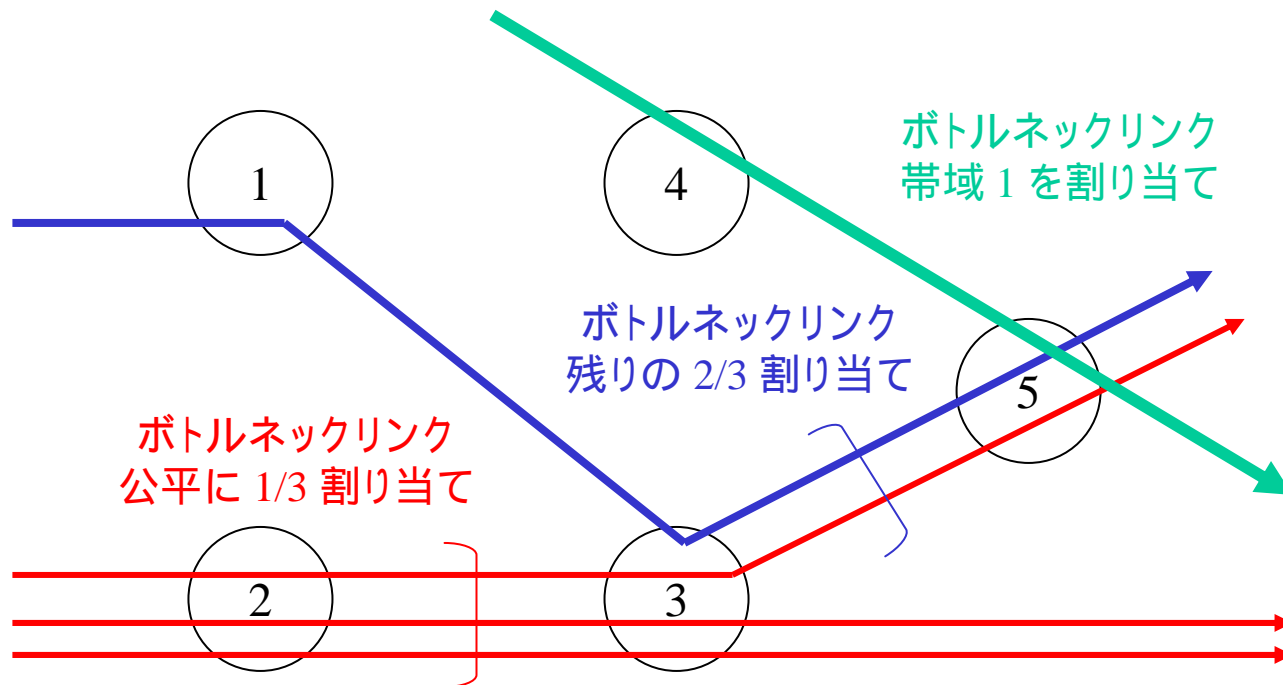
- ACK の受信間隔 (ボトルネック速度) に合わせてパケットを送信



最小最大フロー制御

- 最も少ない帯域割り当てを受けているユーザに対し、最大の帯域割り当てを行う動作を、すべてのユーザに対して繰り返す (最小最大公平)。

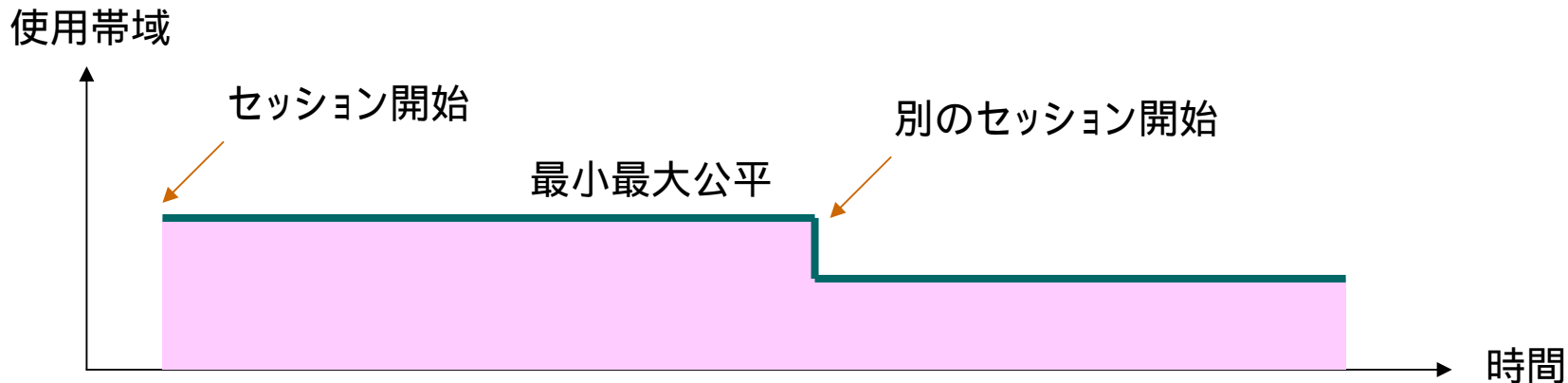
(例) すべてのリンク容量が 1 の場合の以下 (5 セッション) の最小最大公平は？



TCPにおけるフロー制御

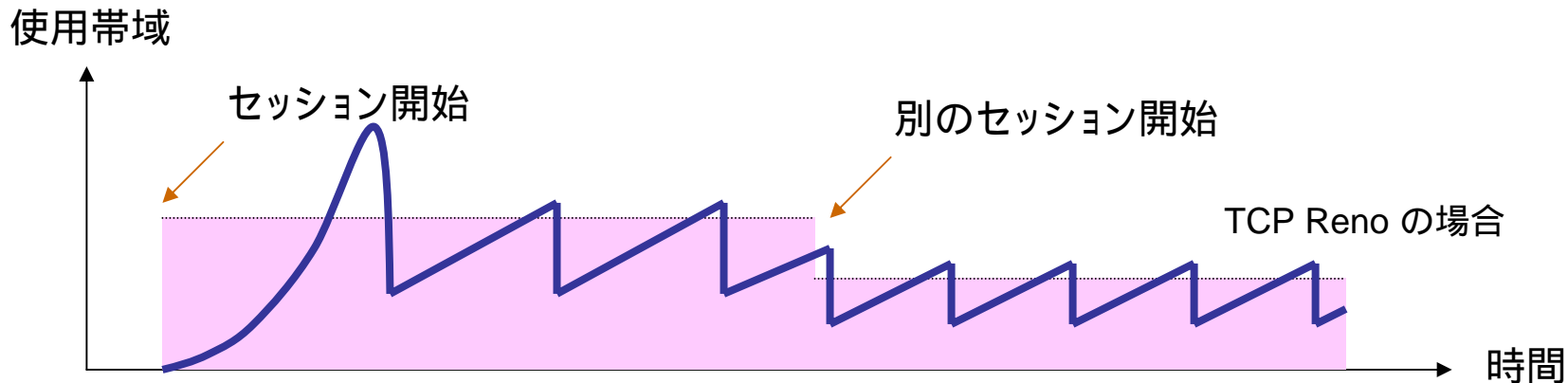
理想:

* 集中型の帯域管理装置 (電話に近い)



TCP: スロースタート + ぶくそう回避

* 端末毎の分散制御



いろいろなTCP

	要点
TCP Tahoe	スロースタート + ぶくそう回避 + 高速再送
TCP Reno	Tahoe + 高速回復
TCP Vegas	RTT (round trip delay) ベースのぶくそう制御
TCP SACK	Reno + 選択的再送 (selective repeat)

- スロースタート: slow start
- ぶくそう回避: congestion avoidance
- 高速再送: fast retransmission
- 高速回復: fast recovery

* 広く用いられているのは TCP Reno

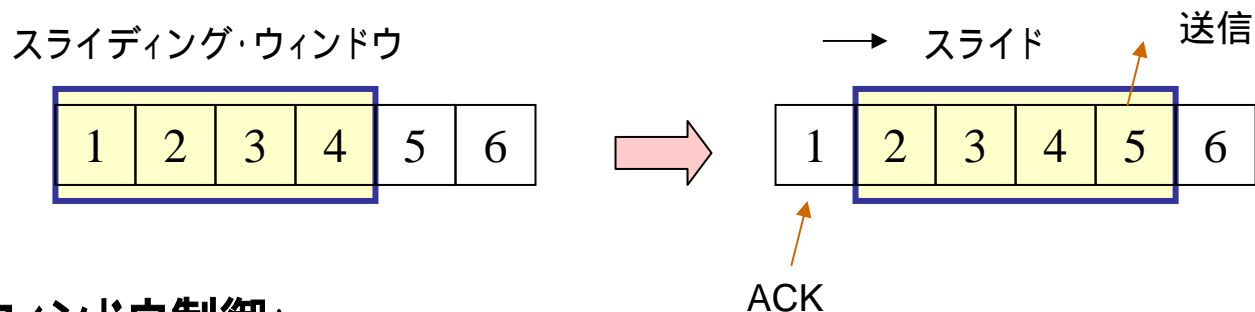
古典的なTCP

- **Go-Back-N ARQ (スライディング・ウィンドウ):**

送信者は ACK を待たずに N 個の packets を送信する

受信者が ACK を返すとウィンドウがスライドして次パケットが送出される

しばしば n 個の packets 毎に1つの ACK を返す (累積応答)



- **ウィンドウ制御:**

rwnd: 広告ウィンドウ (advertisement window)

受信者が要求するセグメント (パケット) サイズ、あるいは受信可能なセグメントサイズを通知し、スライディングウィンドウ (送信パケット数) を制御

欠点: ボトルネックリンクに非常に弱い

TCP Tahoe (1)

- 送信側パラメータを三つ追加:

 cwnd: ぶくそうウィンドウ (congestion window: 初期値1)

 ssthresh: スロースタートとぶくそう回避のモード選択閾値 (初期値大)

 tcprecvthresh: 高速再送を行う重複ACK数 (通常は3)

- スロースタート (指数増加: スループット探索モード):

 if (cwnd < ssthresh)

 --- ACK 毎にパケットを2個送出 ---

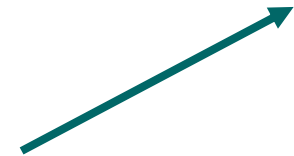
 cwnd += 1;

- ぶくそう回避 (加法増加: スループット安定モード):

 else if (cwnd >= ssthresh)

 --- ACK 毎にパケットを1個送出、cwnd 個送出後1個追加 ---

 cwnd += 1/cwnd;



TCP Tahoe (2)

- 二通りのパケット廃棄の検出:

- (1) 重複 ACK の受信 (TCP ヘッダの ACK ナンバが更新されない場合)
- (2) タイムアウト (ACK が返って来ない場合)

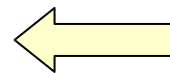
- 高速再送 (軽いふくそう):

ACK が返って来るということは深刻なふくそうではない (仮定)

```
if ( 重複 ACK 数 == tcprecvthresh )
```

```
    --- パケットを再送 ---
```

```
    ssthresh = cwnd/2; cwnd = 1;
```



スロースタートから再開
(ssthresh > cwnd)

- タイムアウト値の更新 (重いふくそう):

タイムアウトが起こるということは深刻なふくそう (仮定)

```
if ( タイムアウト )
```

```
    --- パケットを再送 ---
```

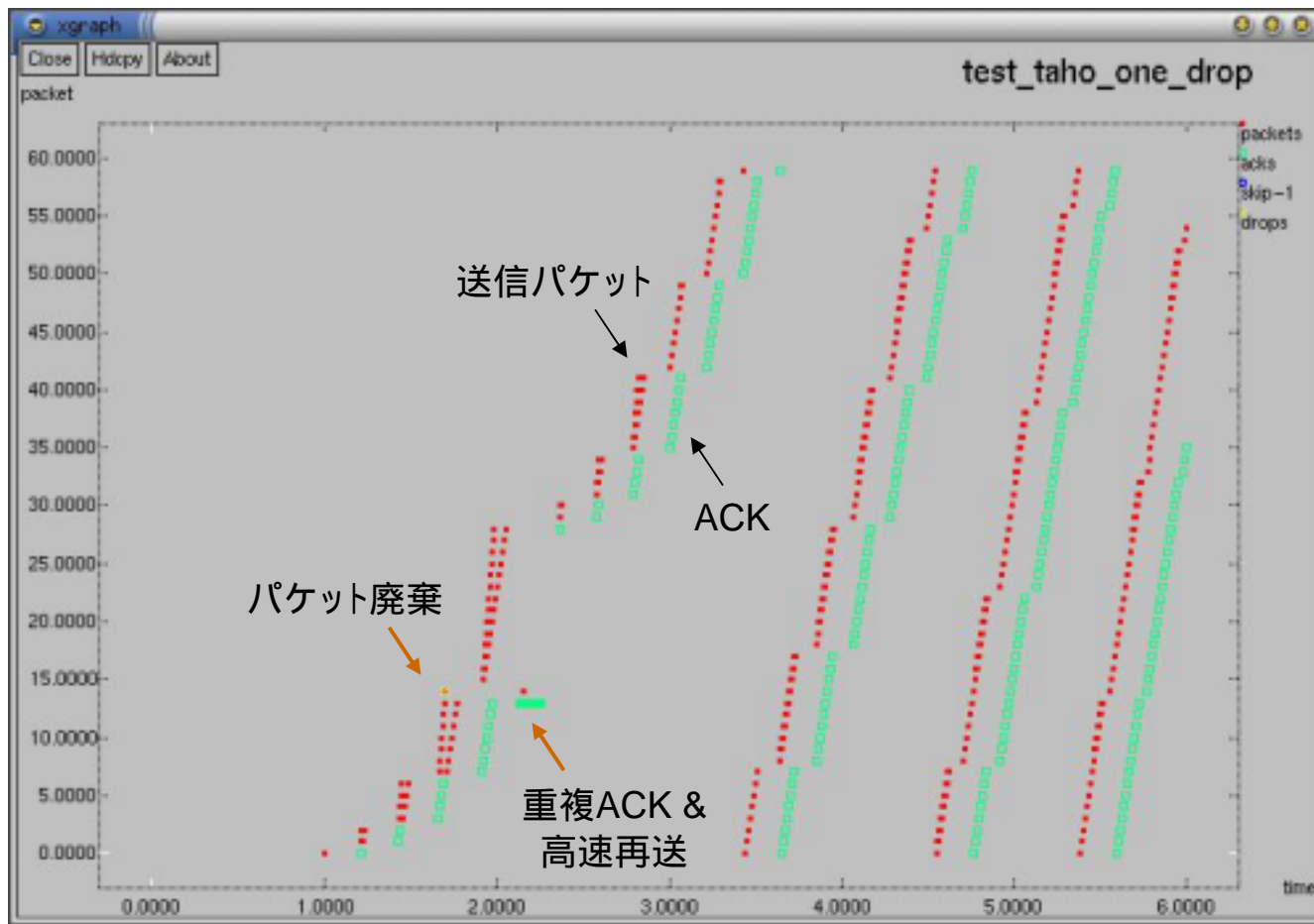
```
    timeout *= 2;
```



指数的バックオフ

TCP Tahoe (3)

パケット数



時間

→
↑ スロースタート(1) ↑ スロースタート(2) ↑ ぶくそう回避

NS (Network Simulator) によるシミュレーション例

TCP Reno (1)

- Tahoe の問題点:

高速再送後、スロースタートに戻る必要は無い

パケット廃棄前の cwnd の値は安全 (仮定: 現在の cwnd の半分)

- 高速回復:

```
if ( 重複 ACK 数 == tcprecvthresh )
```

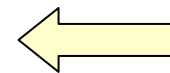
```
--- パケットを再送 (高速再送) ---
```

```
ssthresh = cwnd/2;
```

```
cwnd = cwnd/2 + tcprecvthresh;
```

↑
安全な値

↑
重複 ACK 分 (ACK が正しく返っている)



ふくそう回避モードから再開
(ssthresh < cwnd)

```
if ( 重複 ACK 数 > cwnd/2 )
```

```
--- 重複 ACK 毎に新しいパケットを一つ送信 ---
```



これが妥当な理由を考えよ
(ヒント: cwnd の値が廃棄検出直前の cwnd よりも大きくなる)

```
if ( 再送パケットの確認応答 )
```

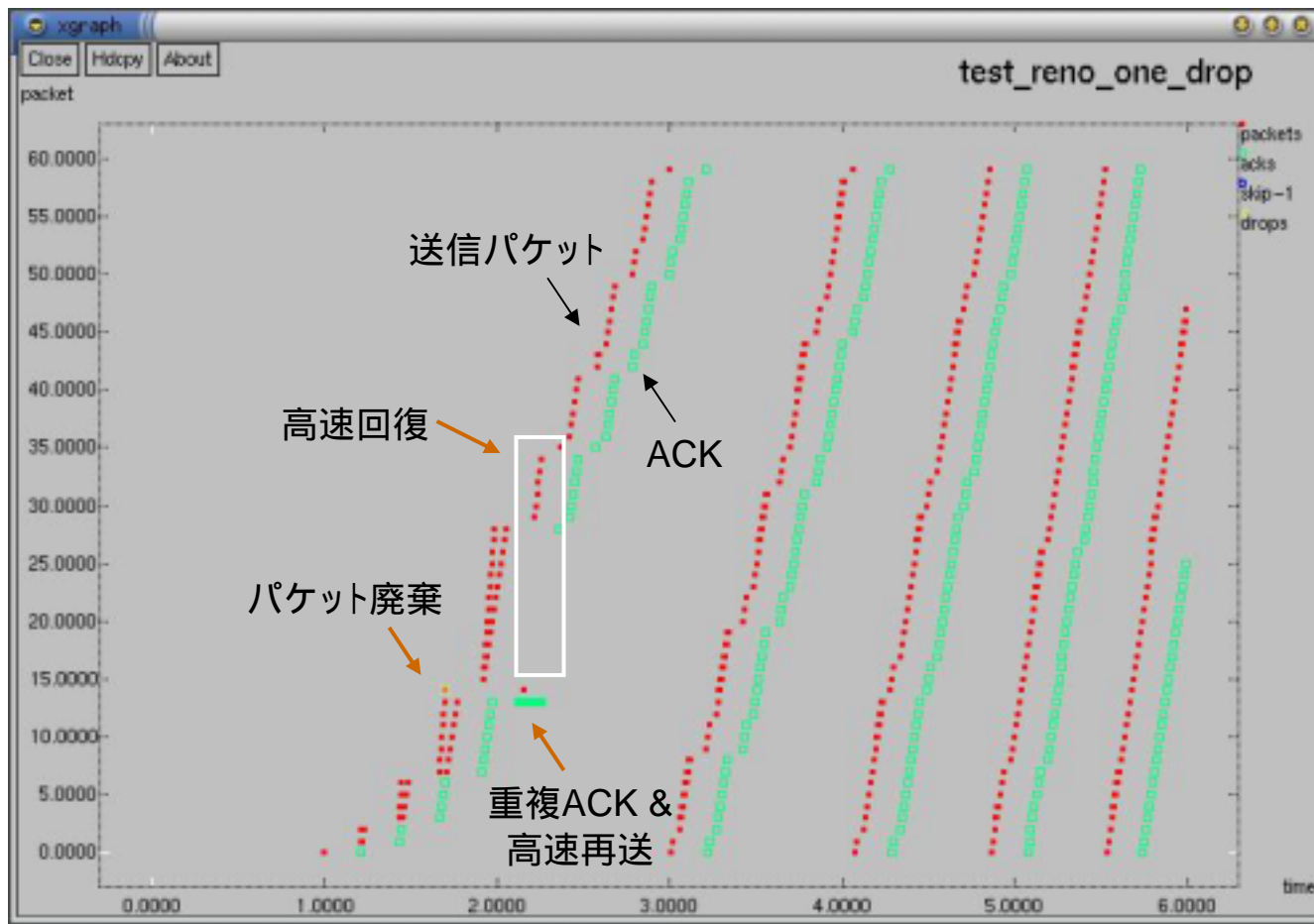
```
cwnd = ssthresh;
```



通常のふくそう回避へ

TCP Reno (2)

パケット数

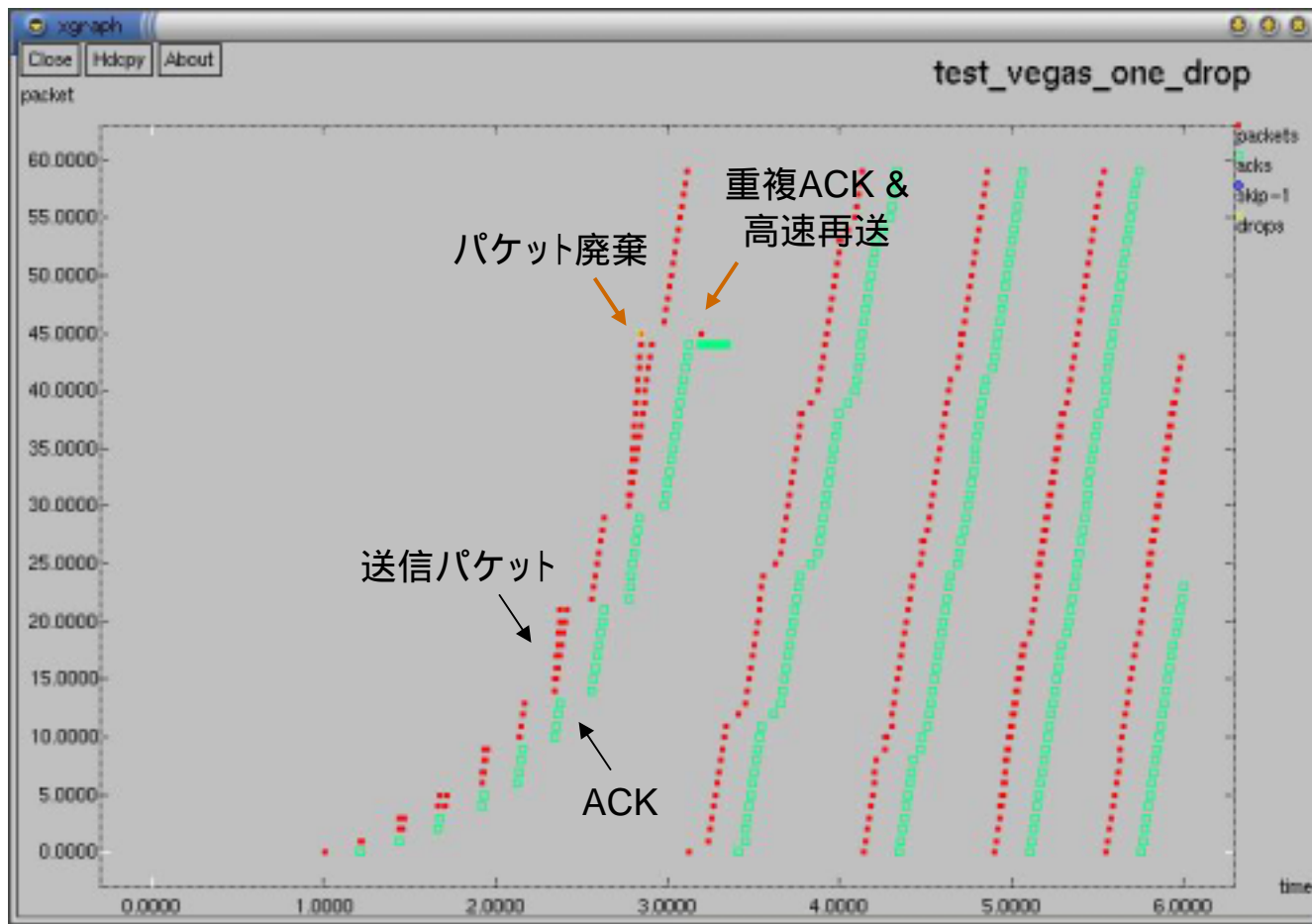


スロースタート 高速回復 ぶくそう回避

NS (Network Simulator) によるシミュレーション例

TCP Vegas (2)

パケット数



時間

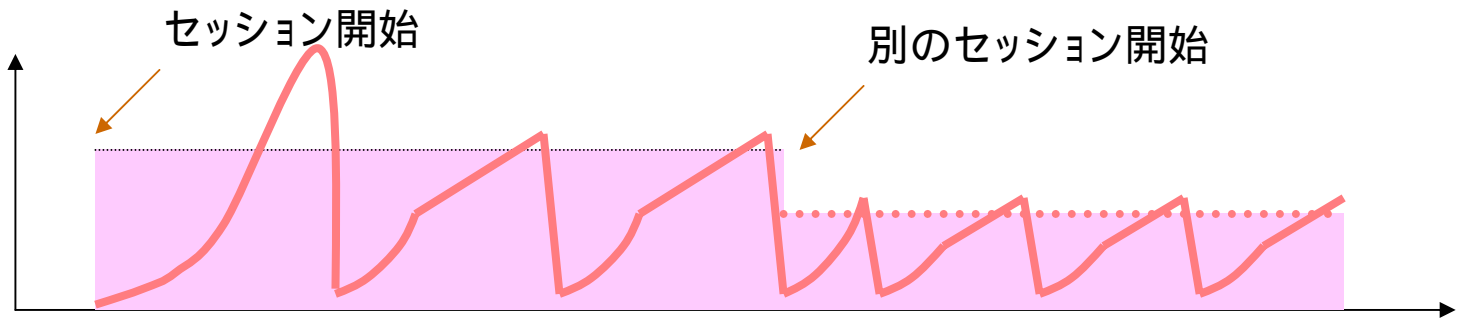
スロースタート

ふくそう回避

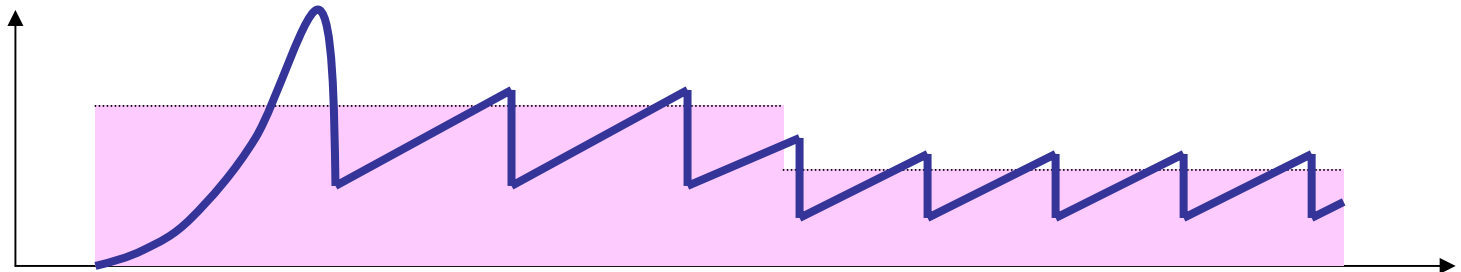
NS (Network Simulator) によるシミュレーション例

直感的な比較

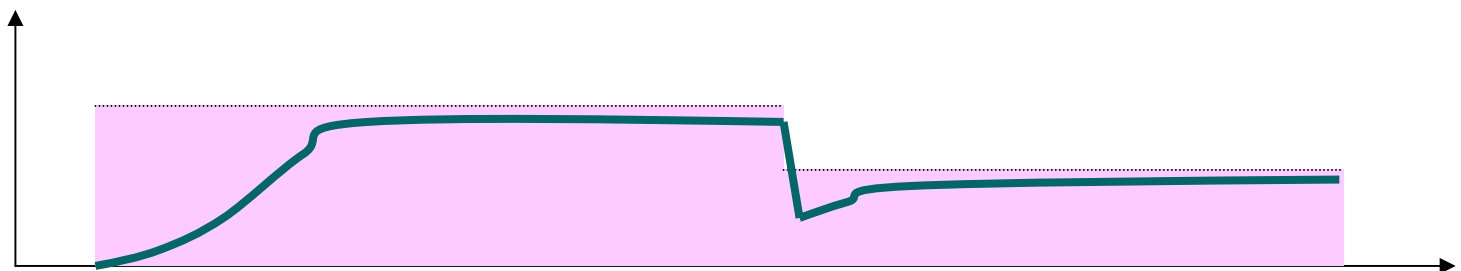
Tahoe



Reno



Vegas



TCPのまとめ

- **再送による信頼性のあるデータ転送:**

反面、転送遅延は増加する。

遅延に敏感なインターネット電話にとっては大きな欠点。遅延が気にならないオンデマンドのインターネット放送では許容範囲。

- **インテリジェントなふくそう制御:**

加法増加 (additive increase) と乗法減少 (multiplicative decrease) を繰り返しながら、それなりのデータ転送速度を実現。

できるかぎり速く送りたいオンデマンドのインターネット放送では望ましい特徴。

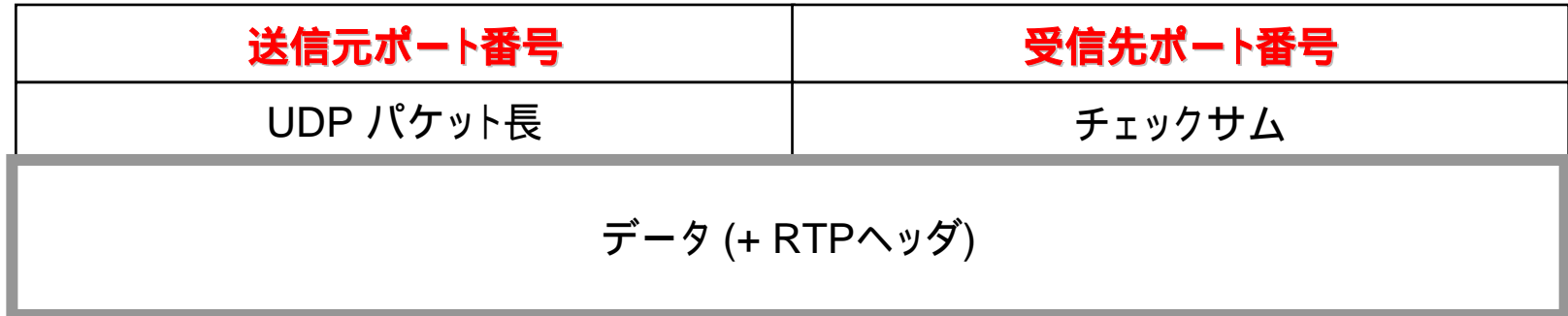
(注) インターネット放送の二形態: ライブ放送とオンデマンド放送

UDP

User Datagram Protocol

UDP ヘッダ

4 byte

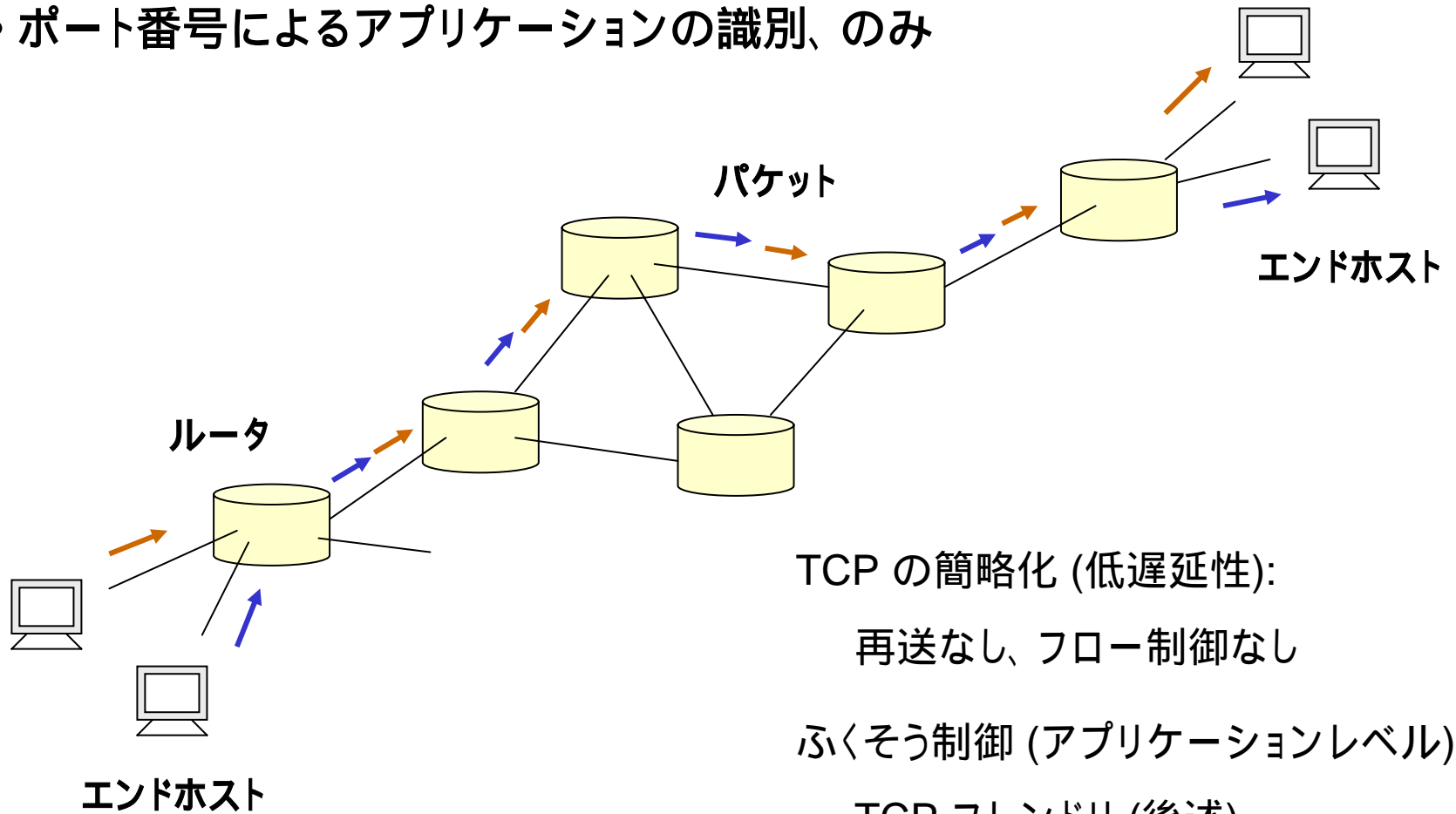


ポート番号: アプリケーションの識別

パケットの紛失、重複、順序逆転などについてまったく関知しない
アプリケーションで対処

UDP の機能

- ポート番号によるアプリケーションの識別、のみ



TCP の簡略化 (低遅延性):

再送なし、フロー制御なし

ふくそう制御 (アプリケーションレベル):

TCP フレンドリ (後述)

低遅延 (UDP) ↔ 信頼性 (TCP)

UDPのまとめ

- **再送を行わない信頼性のないデータ転送:**

転送遅延は抑えられる。

遅延に敏感なインターネット電話にとっては大きな利点。ACK 爆発が発生しないため、マルチキャストにも適している。

- **アプリケーションレベルの誤り制御とふくそう制御 (アダプテーション):**

パケット廃棄やネットワークの輻輳に対して UDP は何も行わないため、アプリケーションレベルで対処する必要がある。

再同期 (パケット廃棄対策)、TCP フレンドリ (輻輳制御)、信頼性マルチキャスト (NACK あるいは FEC)、等

TCP と UDP: まとめ

インターネット電話	TCP	UDP
メディア情報		
制御情報		

インターネット放送	TCP	UDP
オンデマンド放送		
ライブ放送	×	
マルチキャスト	×	(クラスD)
制御情報		(カルーセル)

DCCP

Datagram Congestion Control Protocol

DCCP ヘッダ

4 byte

送信者ポート番号				受信者ポート番号			
オフセット(ヘッダ長)			CCVal	CsCov	チェックサム		
予約	タイプ	X	予約		シーケンスナンバ (high bits)		
(シーケンスナンバ (low bits) ... if X="1")							
(オプション)							
データ							

CCVal (Congestion Control Value) : 別途既定されるプロファイルに応じて、送信者が輻輳制御情報を通知するために使用。

CsCov (Checksum Coverage) : チェックサムがパケットのどの部分までカバーするかを通知。通常 UDP では1ビットでも誤ったらパケットを廃棄するが、CsCov は無線応用を想定してデータ部のビット誤りを許容 ... UDP-Lite (RFC 3828) の考えを踏襲。

DCCP パケット

• タイプフィールドと DCCP パケット

タイプ	DCCPパケット	目的
0	DCCP-Request	クライアントによる接続要求
1	DCCP-Response	接続要求に対するサーバの応答
2	DCCP-Data	アプリケーションデータの送信
3	DCCP-Ack	アプリケーションデータを伴わないACK
4	DCCP-DataAck	アプリケーションデータを伴うACK
5	DCCP-CloseReq	サーバからの接続終了要求
6	DCCP-Close	クライアントからの接続終了要求
7	DCCP-Reset	接続の終了通知
8	DCCP-Sync	障害発生時の同期回復要求
9	DCCP-SyncAck	同期回復要求に対するACK

データの送受信
セットアップ

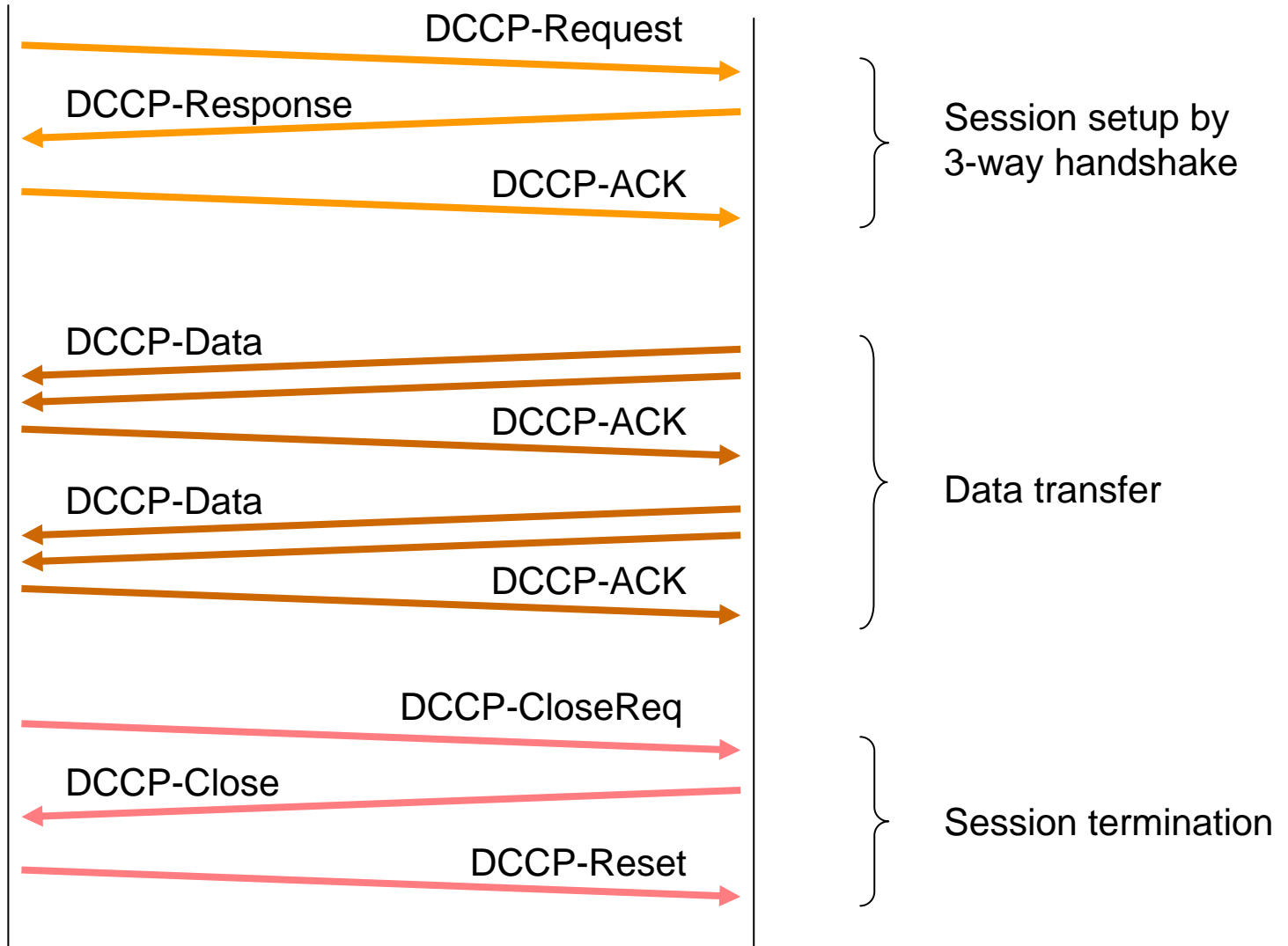
終了処理

障害対策

DCCP 手順

Client

Server



DCCP 輻輳制御

- CCID と DCCP の輻輳制御手段

CCID	輻輳制御手段
0	Reserved
1	Unspecified Sender-based Congestion Control
2	TCP-like Congestion Control (TCP と同様の AIMD 制御: オンラインゲーム用)
3	TFRC Congestion Control (TCPフレンドリ制御: AV系アプリケーション用)

CCID=2: 2個のパケットを受信するたびにACKを返す AIMD制御

CCID=3: RTTごとにACKを返す TFRC制御 (後日、説明)