画像情報特論 (2)

ハイブリッドTCP

情報理工学専攻　甲藤二郎

E-Mail: katto@waseda.jp

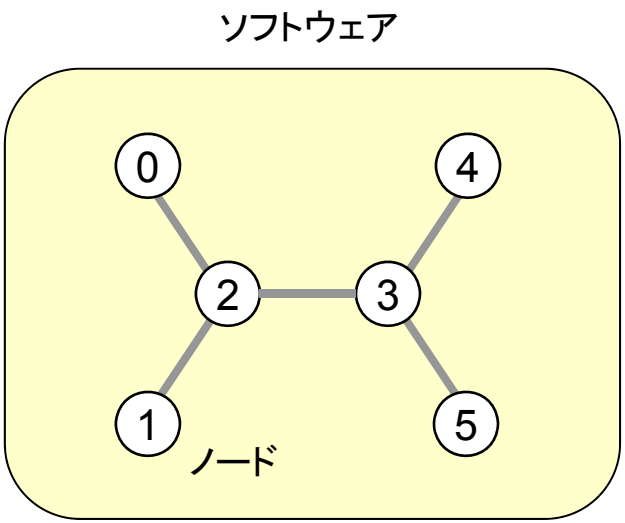
ネットワーク・シミュレーション &
エミュレーション

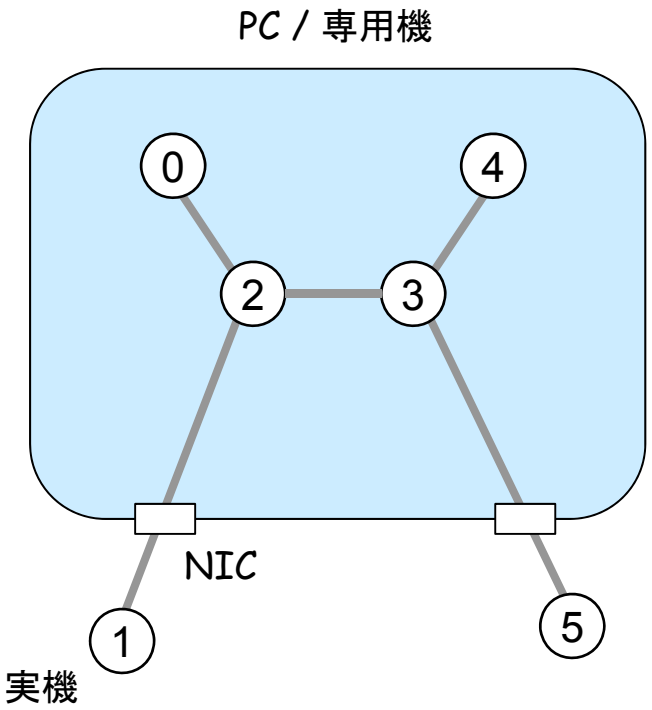
ネットワークの研究

- アルゴリズム
- 理論解析
- シミュレーション
- エミュレーション
- 実装

シミュレータとエミュレータ (1)

- シミュレーション
- エミュレーション



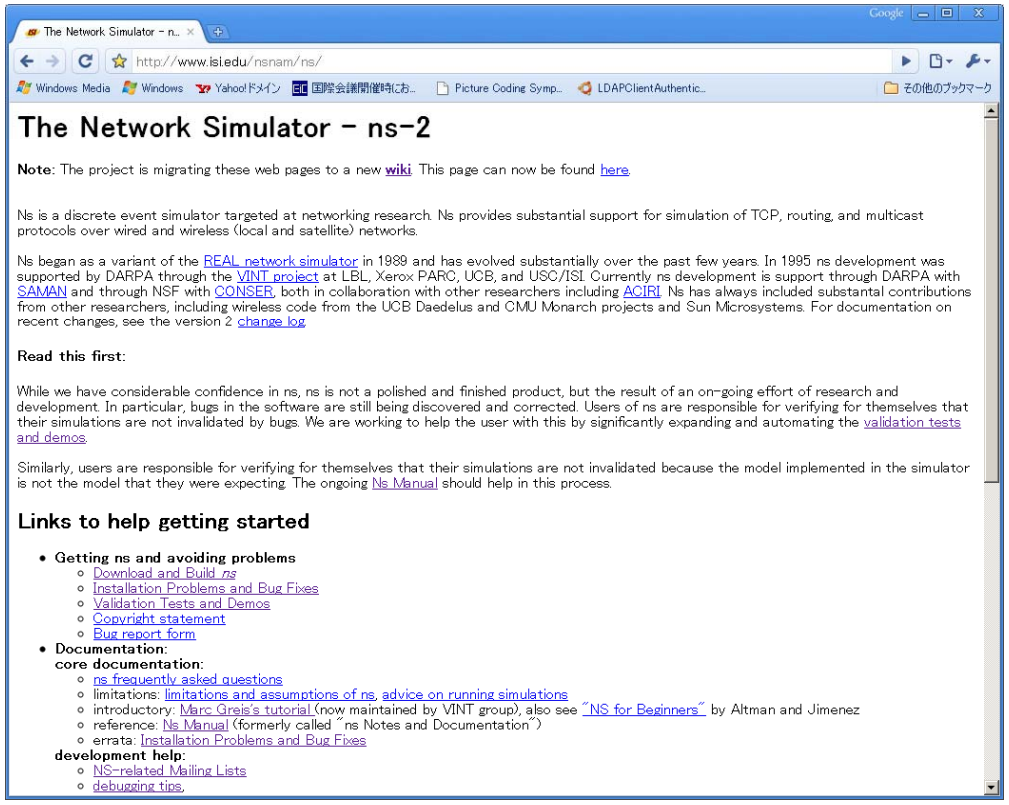


シミュレータとエミュレータ (2)

シミュレータ	エミュレータ	URL
ns-2 (ns)	(nse)	http://www.isi.edu/nsnam/ns/
ns-3		http://www.nsnam.org/
OPNET		http://www.opnet.com/
Qualnet, GloMoSim	EXata	http://www.scalable-networks.com/
Scenergie		http://www.spacetime-eng.com/
	PacketStorm	http://www.packetstorm.com/
	Cloud	http://www.shunra.com/ve-cloud.php

Ns-2 (1)

- http://www.isi.edu/nsnam/ns/



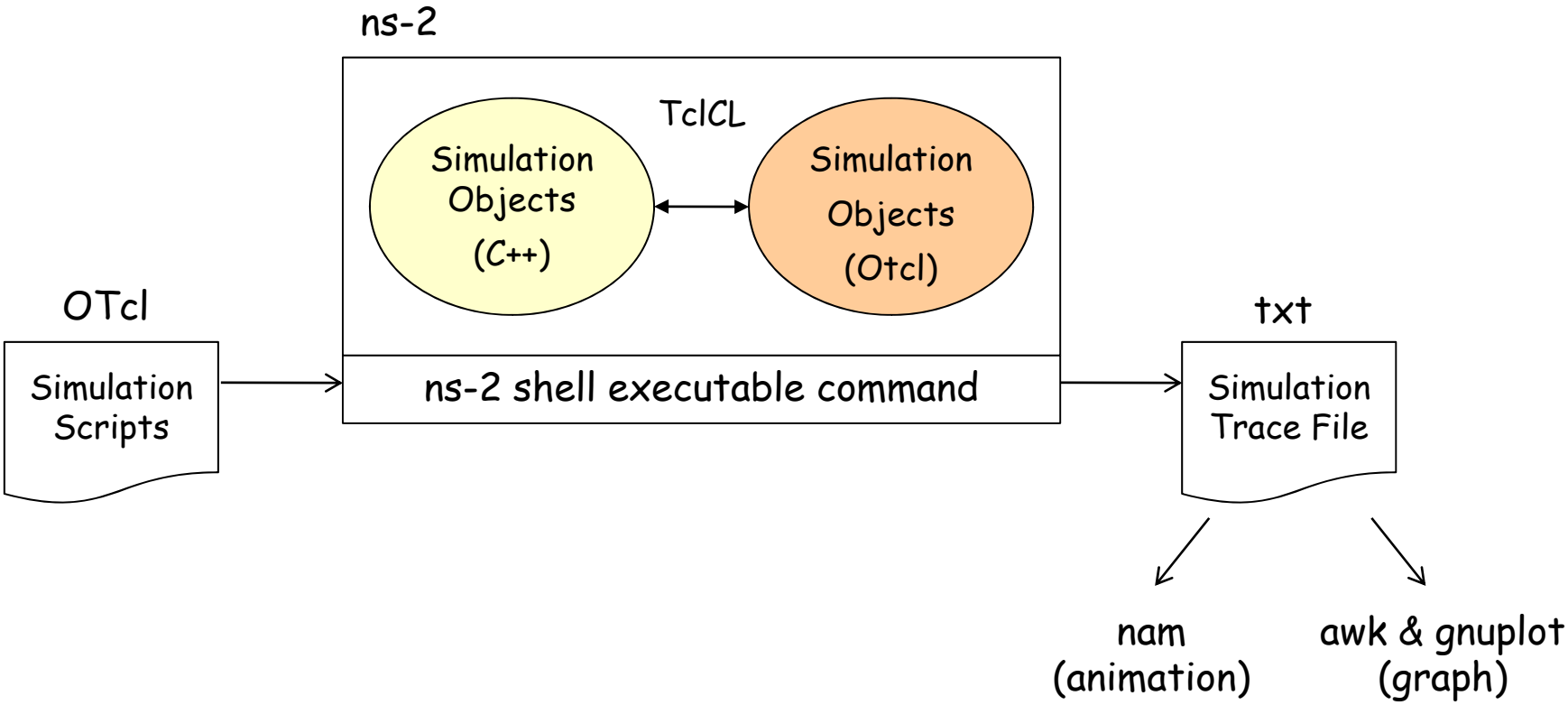
Ns-2 (2)

- ダウンロードサイト
  - 2.29以降:
    http://sourceforge.net/projects/nsnam/
  - 2.28以前:
    http://www.isi.edu/nsnam/dist/

allinone をダウンロード、展開、configure、makeするのが楽
(Tcl/Tk, Otcl, TclCL, ns, nam)

Ns-2 (3)

- ns-2 Architecture



Ns-2 (4)

- Simulation scripts (*.tcl)

```
# パラメータの初期設定
# Simulatorオブジェクトの生成
set ns [ new Simulator ]
# ネットワークトポロジの定義
# エージェントやアプリケーションの定義
# プロシージャの定義 (finish等)
proc finish () …
# イベント定義
$ns at 1.0 "$ftp start"
# シミュレーション開始
$ns run
```

```
set ns [new Simulator]
set f [open out.tr w]
$ns trace-all $f

set n0 [$ns node]
set n1 [$ns node]
$ns duplex-link $n0 $n1 100Mb 1ms DropTail

set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 attach-agent $udp0
…
```

Ns-2 (5)

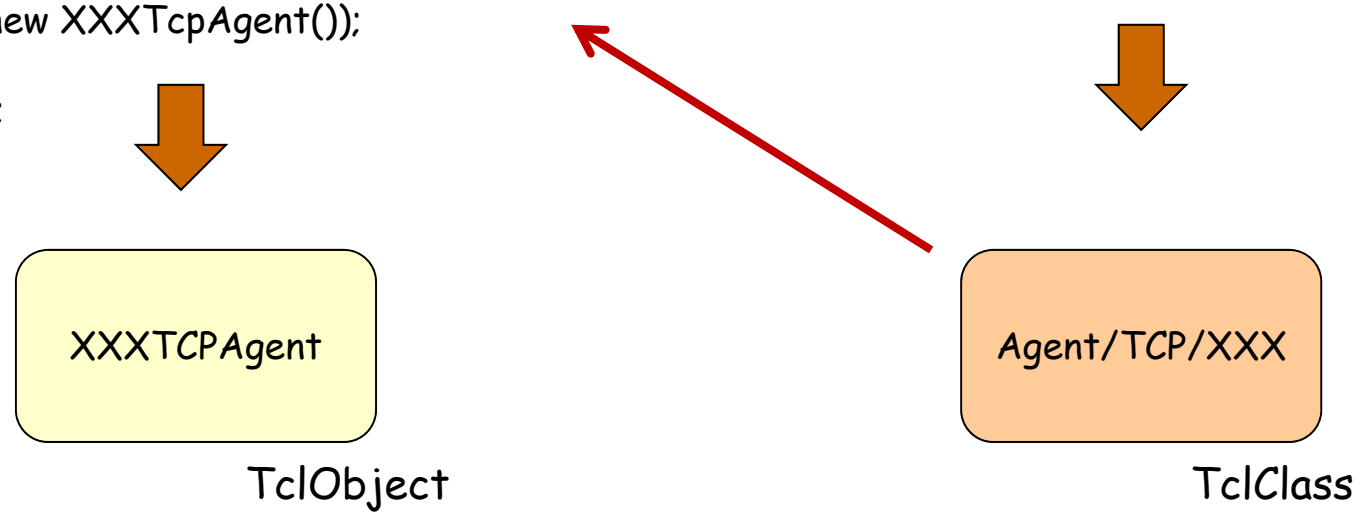
- Simulation Objects (C++/OTcl)

C++

```
static class XXXTcpClass : public TclClass {
public:
   XXXTcpClass() : TclClass("Agent/TCP/XXX") {}
   TclObject* create(int, const char*const*) {
      return (new XXXTcpAgent());
   }
} class_XXX;
```

OTcl

```
set tcp [new Agent/TCP/XXX]
```



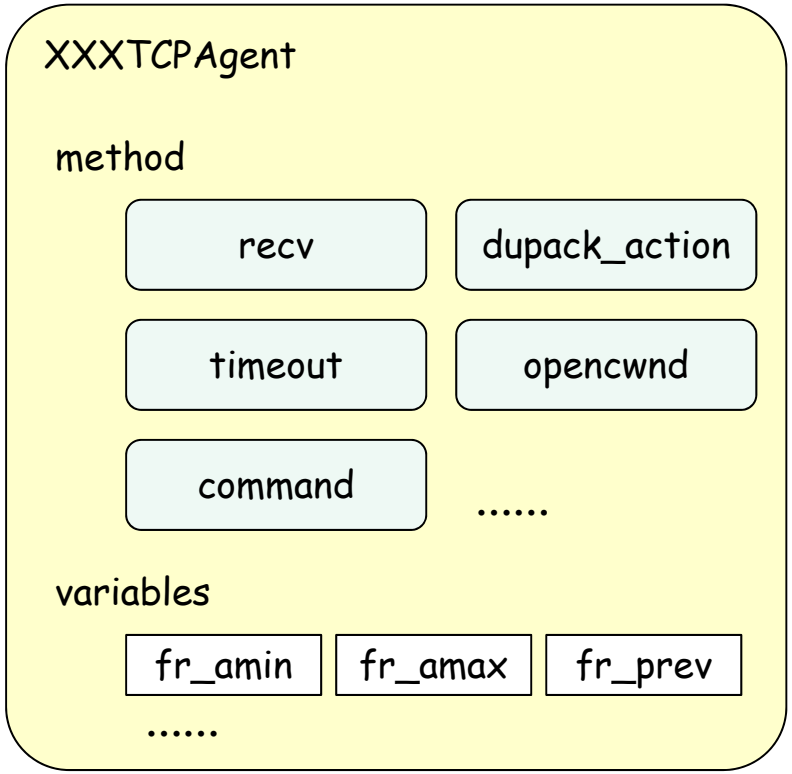
Ns-2 (6)

- Simulation Objects (C++/OTcl)

C++

```
class XXXTcpAgent : public TcpAgent {
public:
  XXXTcpAgent();
  virtual void recv(Packet *pkt, Handler*);
  virtual void dupack_action();
  virtual void timeout (int tno);
  virtual void opencwnd();
  …
protected:
  int command(int argc, const char*const* argv);

  double fr_amin_;
  double fr_amax_;
  double fr_prev_;
}
```



Ns-2 (7)

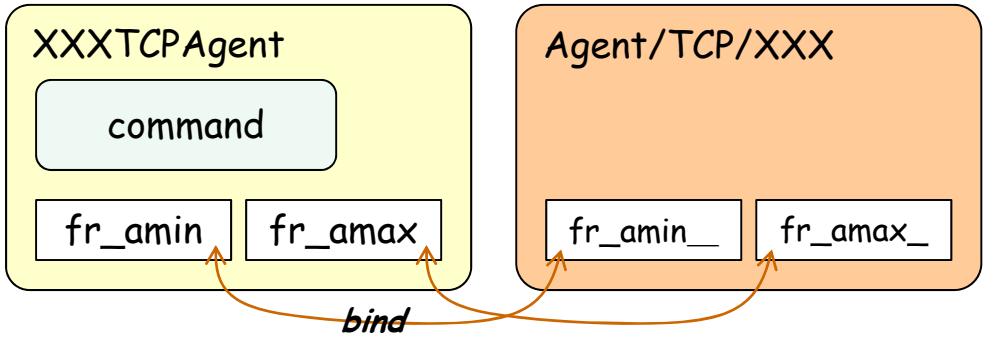
- Simulation Objects (C++/OTcl)

C++

```
XXXTcpAgent::XXXTcpAgent()
{
  bind("fr_amin_", &fr_amin_);
  bind("fr_amax_", &fr_amax_);
  ...
}
XXXTCPAgent::command(int argc, const char*const* argv)
{
  if  (argc == 3) {
    if ( strcmp(argv[1], "target") == 0 ) {
      ...
    }
  }
  return (NsObject::command(argc,argv));
}
```

OTcl

```
set tcp [new Agent/TCP/XXX]
$tcp set fid_ 1
$tcp set fr_amin_ 0.2
$tcp set fr_amax_ 0.8
$tcp target [new Agent/Null]
```



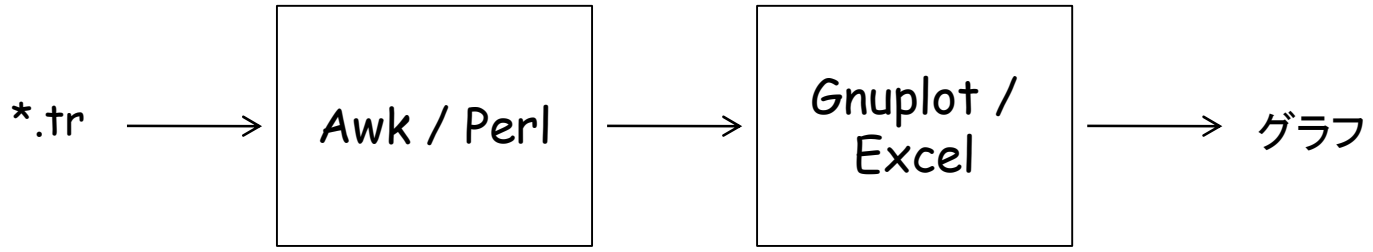
Ns-2 (8)

- Trace File (*.tr)


```
+ 1.84375 0 2 cbr 210 ------- 0 0.0 3.1 225 610
- 1.84375 0 2 cbr 210 ------- 0 0.0 3.1 225 610
r 1.84471 2 1 cbr 210 ------- 1 3.0 1.0 195 600
r 1.84566 2 0 ack 40 ------- 2 3.2 0.1 82 602
+ 1.84566 0 2 tcp 1000 ------- 2 0.1 3.2 102 611
- 1.84566 0 2 tcp 1000 ------- 2 0.1 3.2 102 611
r 1.84609 0 2 cbr 210 ------- 0 0.0 3.1 225 610
+ 1.84609 2 3 cbr 210 ------- 0 0.0 3.1 225 610
d 1.84609 2 3 cbr 210 ------- 0 0.0 3.1 225 610
- 1.8461 2 3 cbr 210 ------- 0 0.0 3.1 192 511
r 1.84612 3 2 cbr 210 ------- 1 3.0 1.0 196 603
```

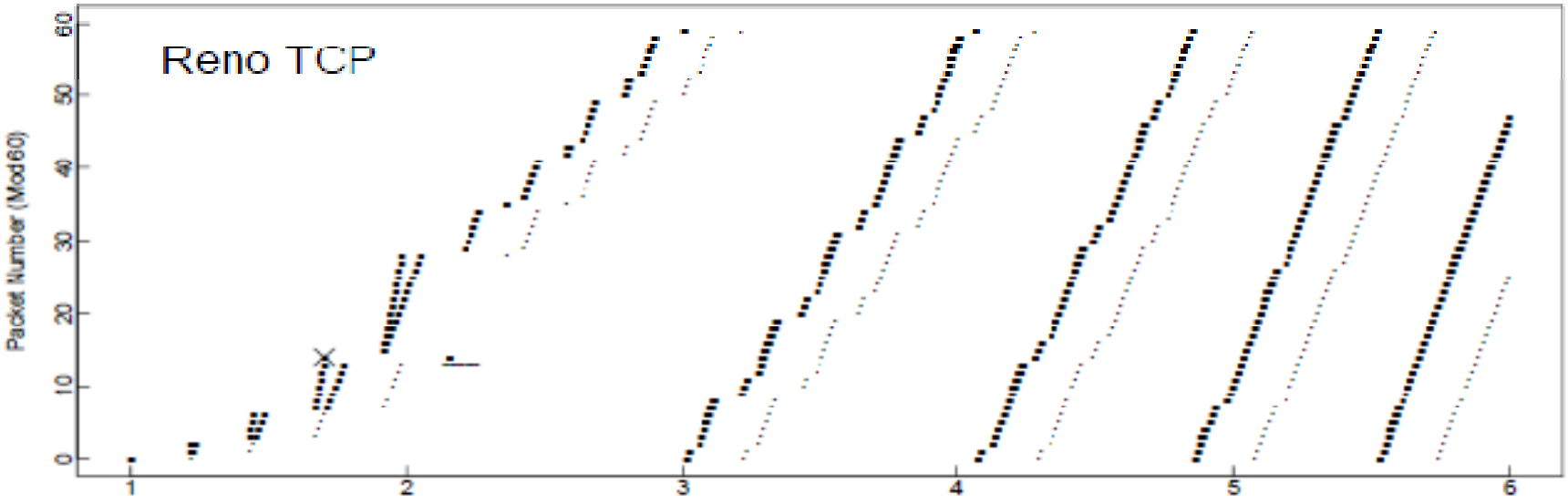
Ns-2 (9)

- Awk / Perl
  - スクリプトを用いたトレースファイルの整形
  - Gnuplotなどに合わせた出力



Ns-2 (10)

・結果の例



電子情報通信学会ネットワークシステム研究会アーカイブ

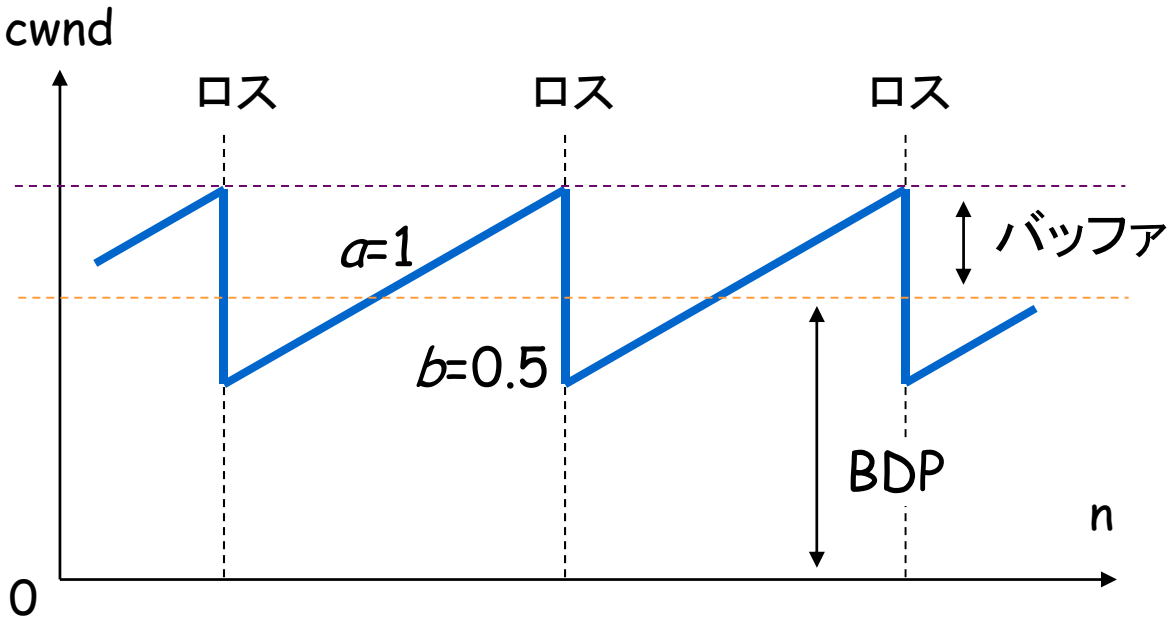
- http://www.ieice.org/~ns/jpn/archives.html
  - 2009/8: ns-2 (サマースクール)
  - 2009/8: OPNET (サマースクール)
  - 2009/12: Qualnet (チュートリアル)

その他、「ns-2 チュートリアル」などで検索をかければ情報多数

※ ns-3の情報はまだ少ない ⇒ チャンス

ハイブリッドTCP

TCP-Reno (loss-driven)

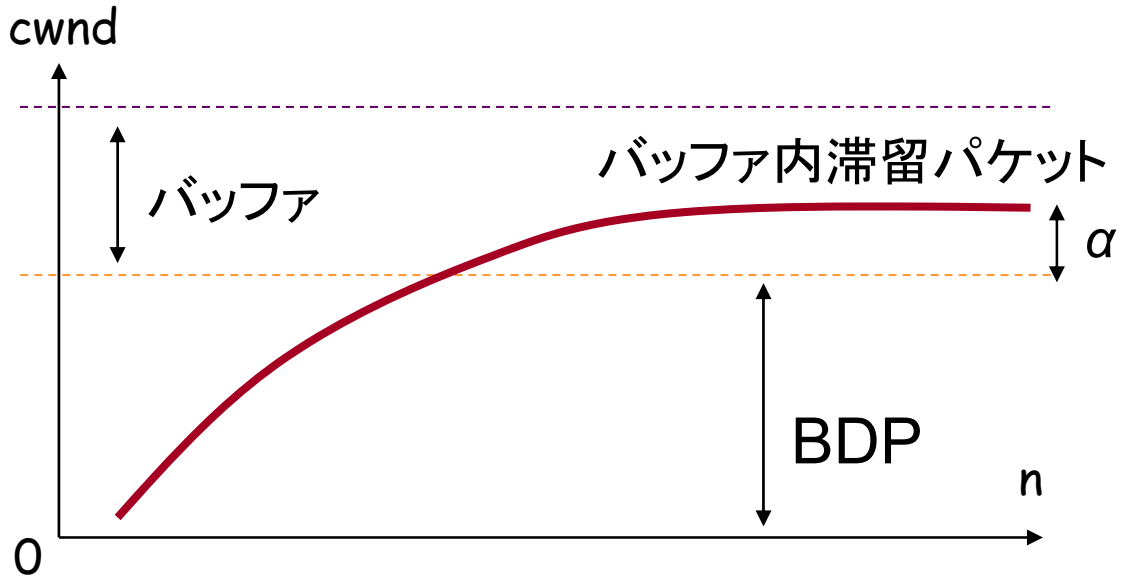


increase: cwnd = cwnd + 1/cwnd

decrease: cwnd = cwnd / 2

AIMD: additive increase multiplicative decrease

TCP-Vegas (delay-driven)



e.g. α=1, β=3

$$diff = \left(\frac{cwnd}{RTT_{\min}} - \frac{cwnd}{RTT} \right) \cdot RTT_{\min}$$

バッファ内滞留パケット数

increase: $$cwnd = \begin{cases} cwnd + 1 & diff < \alpha \\ cwnd & otherwise \\ cwnd - 1 & diff > \beta \end{cases}$$

decrease: $$cwnd = cwnd * 0.75$$

10年前のTCPの課題

- 有線ネットワークの高速化
  - ウィンドウの増加が追い付かない（非効率性）
- 無線ネットワークの普及
  - 無線エラーとバッファオーバーフローを区別できない

- TCP-Reno (NewReno, SACK) の支配
  - Reno は Vegas を排除する（非親和性）

2000年代の TCP Variants

- **Loss-driven (AIMD)**
  - TCP-Reno / NewReno / SACK
  - High-Speed TCP (IETF RFC 3649, Dec 2003)
  - Scalable TCP (PFLDnet 2003)
  - BIC-TCP / CUBIC-TCP (IEEE INFOCOM 2004, PFLDnet 2005)
  - H-TCP (PFLDnet 2004)
  - TCP-Westwood (ACM MOBICOM 2001)
- **Delay-driven (RTT Observation)**
  - TCP-Vegas (IEEE JSAC, Oct 1995)
  - FAST-TCP (INFOCOM 2004)
- **Hybrid**
  - Gentle High-Speed TCP (PfHSN 2003)
  - TCP-Africa (IEEE INFOCOM 2005)
  - Compound TCP (PFLDnet 2006)
  - Adaptive Reno (PFLDnet 2006)
  - YeAH-TCP (PFLDnet 2007)
  - TCP-Fusion (PFLDnet 2007)

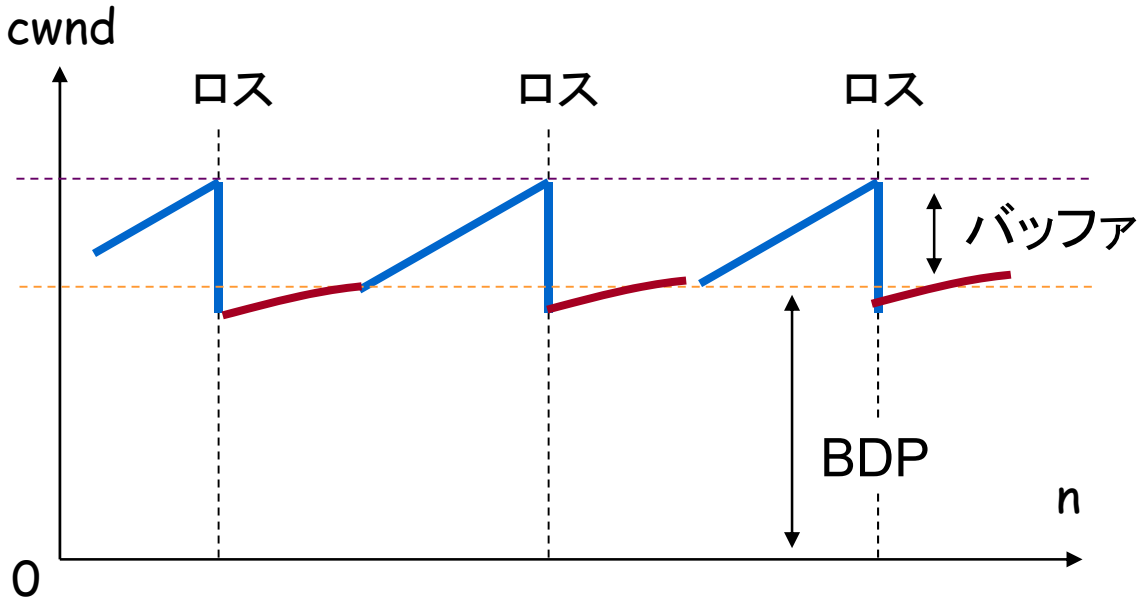
Loss-driven TCPs

		a	b
	Variants	**Increase / Update**	**Decrease**
	TCP-Reno	1	0.5
aggressive	HighSpeed TCP (HS-TCP)	$a(w) = \frac{2w^2 \cdot b(w) \cdot p(w)}{2 - b(w)}$ e.g. 70 (10Gbps, 100ms)	$b(w) = \frac{\log(w) - \log(W_{low})}{\log(W_{high}) - \log(W_{low})}(b_{high} - 0.5) + 0.5$ e.g. 0.1 (10Gbps, 100ms)
aggressive	Scalable TCP (STCP)	0.01 (per every ACK)	0.875
adaptive	BIC-TCP	$\begin{cases} additive\ increase\ (fast) \\ binary\ search\ (slow) \\ max\ probing\ (fast) \end{cases}$	0.875
adaptive	CUBIC-TCP	$w = 0.4(t - \sqrt[3]{2W_{\max}})^3 + W_{\max}$	0.8
adaptive	H-TCP	$\alpha \leftarrow 2(1-\beta)\{1 + 10.5 \cdot (t - TH)\}$	$\beta \leftarrow \begin{cases} 0.5 & for\ \left\lvert \frac{B(k+1) - B(k)}{B(k)} \right\rvert > 2 \\ \frac{RTT_{\min}}{RTT_{\max}} & otherwise \end{cases}$
adaptive	TCP-Westwood (TCPW)	1	$\begin{cases} RE * RTT_{\min} / PS & (not\ congested) \\ BE * RTT_{\min} / PS & (congested) \end{cases}$

Delay-driven TCPs

	a	*b*
Variants	**Update**	**Decrease**
TCP-Vegas	$w \leftarrow \begin{cases} w+1 & (\textit{no congestion}) \\ w & (\textit{stable}) \\ w-1 & (\textit{early congestion}) \end{cases}$	0.75
FAST-TCP	$w \leftarrow \min\left\{2w, (1-\gamma)w + \gamma\left(\frac{RTT_{\min}}{RTT}w + \alpha\right)\right\}$	0.5 (?)

Hybrid TCP



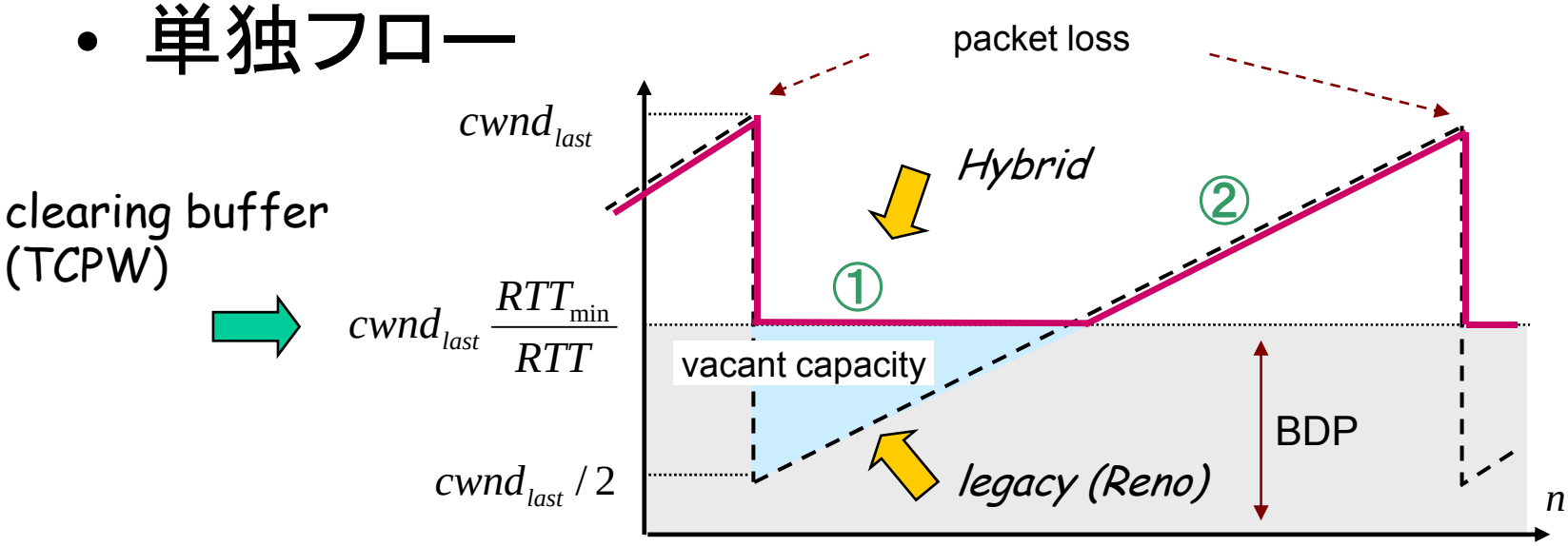
- RTT増加なし: ロスモード ⇒ 親和性の改善
- RTT増加あり: 遅延モード ⇒ 効率性の改善

Hybrid TCPs

	Variants	Increase (a)	Decrease (b)
simple	Gentle HS-TCP	HS-TCP / Reno	HS-TCP
simple	TCP-Africa	HS-TCP / Reno	HS-TCP
adaptive	Compound TCP (CTCP)	$0.125 \cdot cwnd^{0.75}$ / Reno	0.5
adaptive	Adaptive Reno (ARENO)	$B/10\text{Mbps}$ / Reno	$\begin{cases} 1 & (\textit{non congestion loss}) \\ 0.5 & (\textit{congestion loss}) \end{cases}$
adaptive	YeAH-TCP	STCP / Reno	$\max\left(\frac{RTT_{\min}}{RTT}, 0.5\right)$
adaptive	TCP-Fusion	$\frac{B * D_{\min}}{PS}$ / Reno	$\max\left(\frac{RTT_{\min}}{RTT}, 0.5\right)$

Hybrid TCP (1)

- 単独フロー



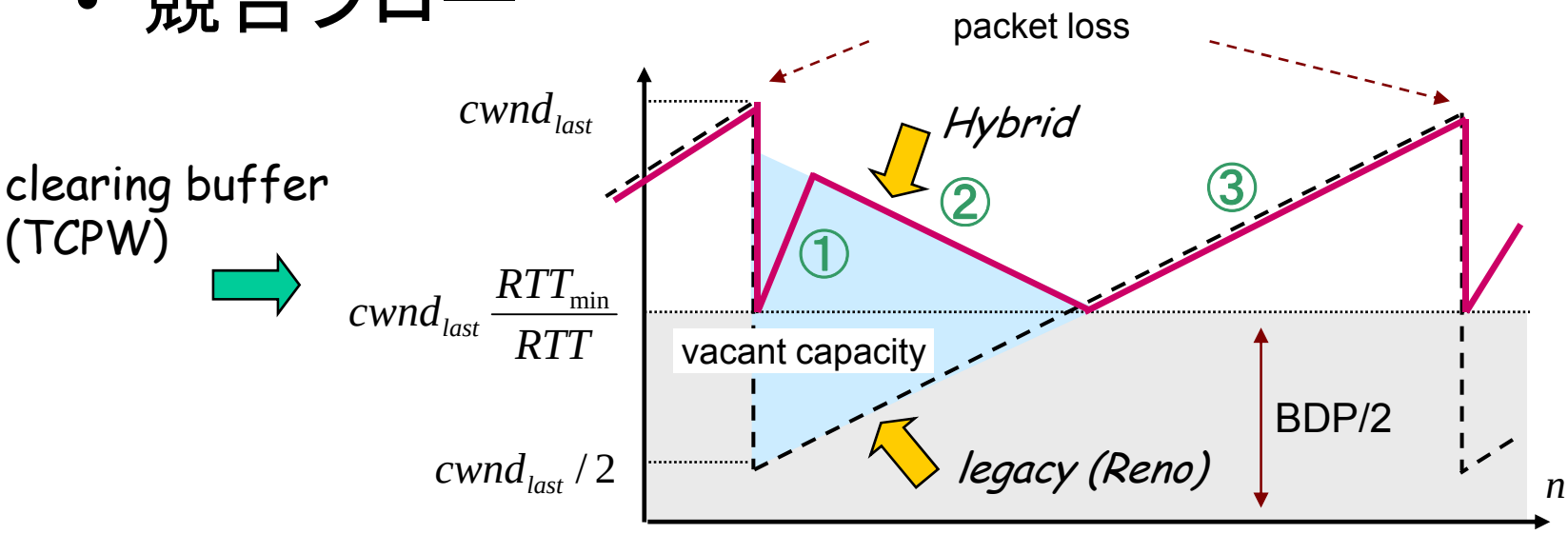
二つのモードの適応的な切り替え (loss & delay):

① RTTが増加するまでは一定レート (delay mode) : 効率性

② RTTが増加し始めたら Reno として動作 (loss mode) : 親和性

Hybrid TCP (2)

- 競合フロー



二つのモードの適応的な切り替え (loss & delay):

① cwnd の急激な増加 (delay ... 効率性)

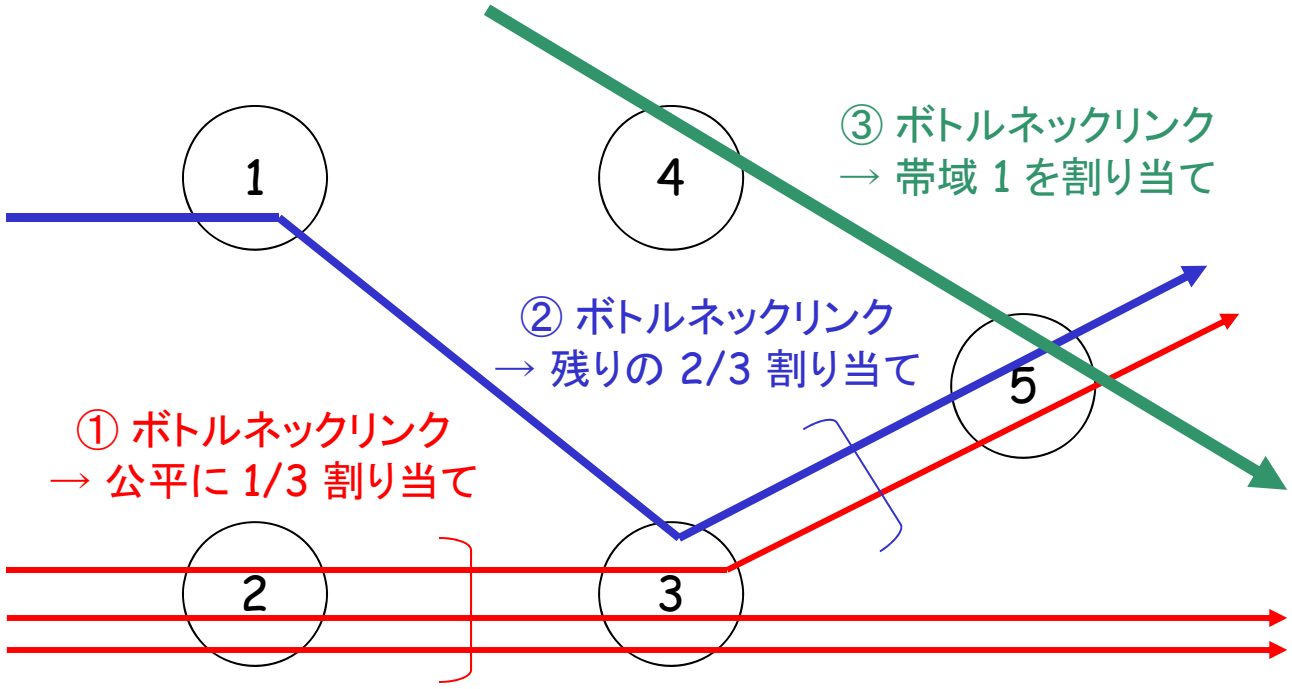
② cwnd のなだらかな減少 (delay ... 輻輳回避)

③ RTTが増加し始めたら Reno として動作(loss ... 親和性)

ハイブリッドTCPの特性解析

Min-Max Fair (理想解)

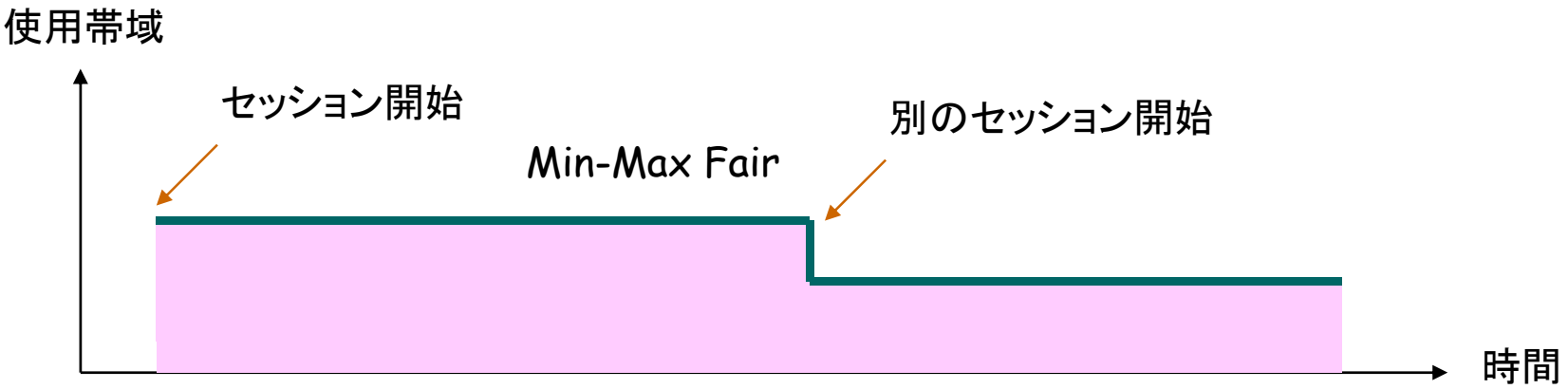
- 最も少ない帯域割当てを受けているユーザに最大の帯域割当てを行う動作を、すべてのユーザに対して繰り返す (<u>最小最大公平</u>)。



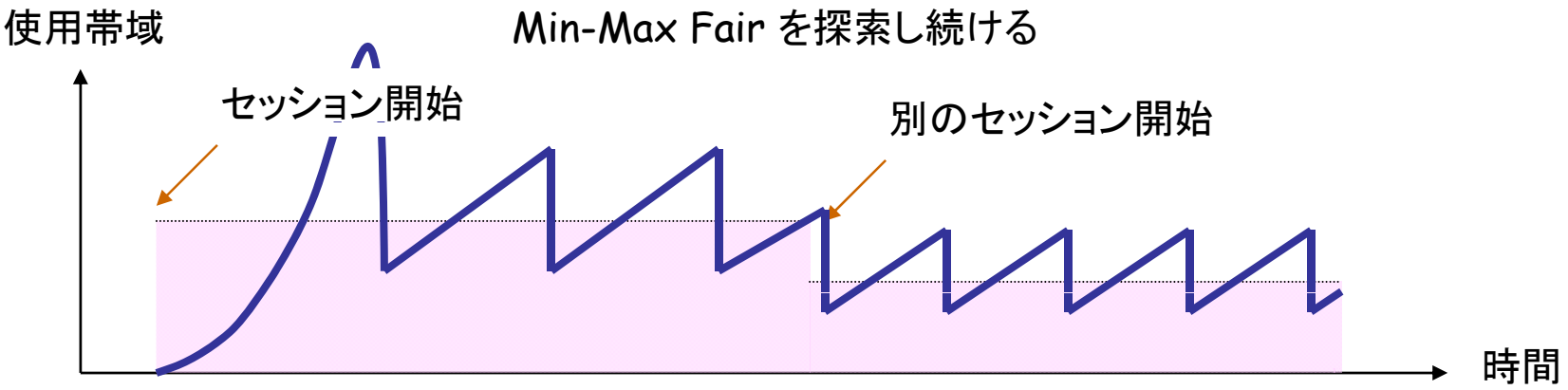
D.Bertsekas and R.Gallager: "Data Networks," Prentice Hall.

TCPの目標

理想解:



TCP Reno



TCPの理想挙動モデル (1)

- モデルの定義
  - Loss-driven (TCP-Reno) :
    - cwnd += 1 (“RTTラウンド” 毎)
    - cwnd *= 1/2 (パケットロス時)
  - Delay-driven :
    - RTTが増加しないように “パイプ” を埋める(リンクを使い切る)
  - Hybrid :
    - RTTが増加しない場合は delay モードで動作
    - RTTが増加し始めたら loss モードで動作
    - モード選択: cwnd = max($cwnd_{loss}$, $cwnd_{delay}$)

TCPの理想挙動モデル (2)

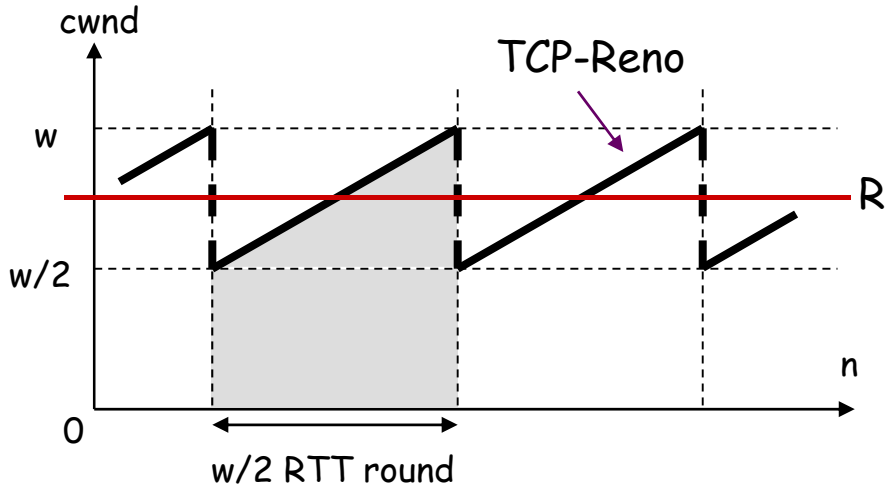
- パラメータの定義
  - w : パケットロスが発生するパケット数
  - W : ちょうどリンクを使い切るパケット数 ～ BDP
  - p : パケットロス率

- (乱暴な) 仮定
  - バッファオーバーフローによるパケットロスと、ランダムエラーによるパケットロスの影響を等価とみなす。

$$p = \frac{8}{3w^2}$$ (in case of TCP-Reno)

TCPの理想挙動モデル (3)

- TCPフレンドリのモデル



w: パケットロスが起きる cwnd
p: パケットロス率
RTT: ラウンドトリップ遅延

R: 等価なレート

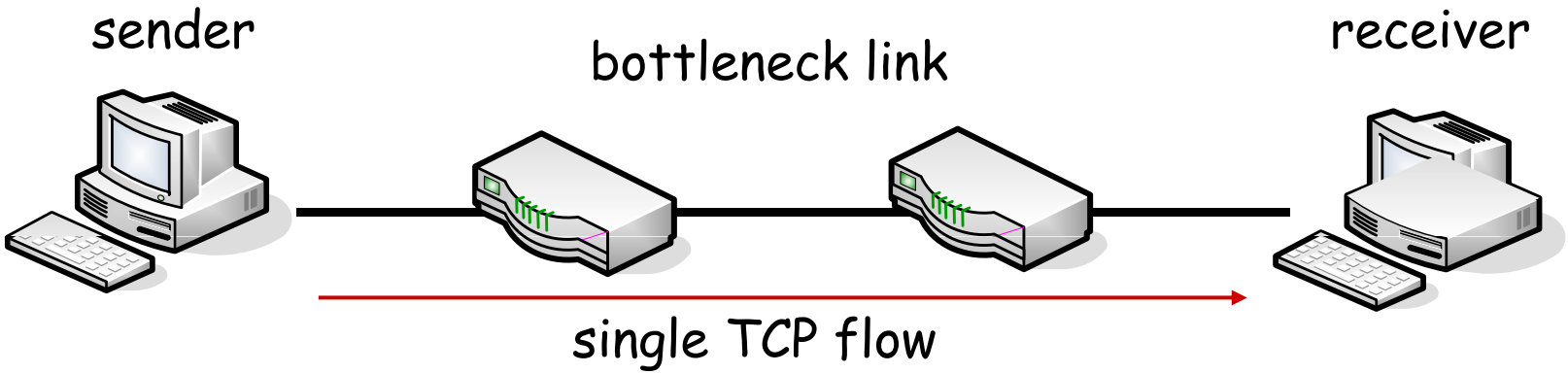
パケットロスが起きるまで送れるパケット数(= 台形の面積)

$$\frac{1}{2}\cdot\left(\frac{w}{2}+w\right)\cdot\frac{w}{2}=\frac{3w^2}{8}$$

$$\begin{cases} p=\dfrac{8}{3w^2} \\ R=\dfrac{PS}{RTT}\cdot\sqrt{\dfrac{3}{2p}} \end{cases}$$

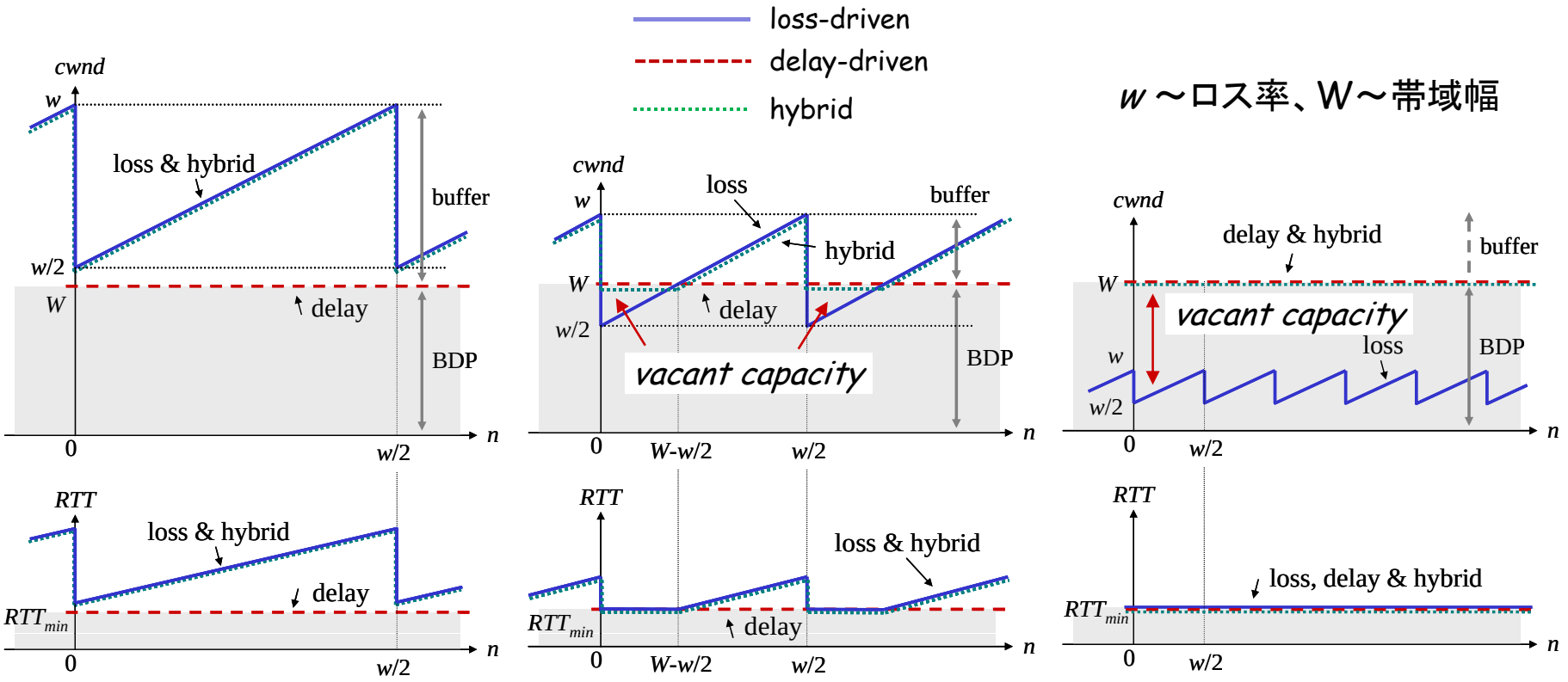
TCPの理想挙動モデル (4)

- 単独フローの場合



TCPの理想挙動モデル (5)

- cwnd と RTT の挙動



(i) $W < w/2$

バッファ大、ロス率小
(常にloss-mode)

(ii) $w/2 < W < w$

バッファ小、ロス率中
(delay → loss)

(iii) $w < W$

ロス率大、常に空き帯域
(常にdelay-mode)

TCPの理想挙動モデル (6)

- 定式化

TCP	CA round	(i) $W < w/2$	(ii) $w/2 \le W < w$	(iii) $w \le W$
Loss	transmitted packets	$\frac{3}{8}w^2$	$\frac{3}{8}w^2$	$\frac{3}{8}w^2$
	elapsed time	$\frac{1}{2}w \cdot RTT_{\min} + \frac{1}{8}(3w^2 - 4wW) \cdot \frac{PS}{B}$	$\frac{1}{2}w \cdot RTT_{\min} + \frac{1}{2}(w-W)^2 \cdot \frac{PS}{B}$	$\frac{1}{2}w \cdot RTT_{\min}$
Delay	transmitted packets	$\frac{1}{2}w \cdot W$	$\frac{1}{2}w \cdot W$	$\frac{1}{2}w \cdot W$
	elapsed time	$\frac{1}{2}w \cdot RTT_{\min}$	$\frac{1}{2}w \cdot RTT_{\min}$	$\frac{1}{2}w \cdot RTT_{\min}$
Hybrid	transmitted packets	$\frac{3}{8}w^2$	$\frac{1}{2}w \cdot W + \frac{1}{2}(w-W)^2$	$\frac{1}{2}w \cdot W$
	elapsed time	$\frac{1}{2}w \cdot RTT_{\min} + \frac{1}{8}(3w^2 - 4wW) \cdot \frac{PS}{B}$	$\frac{1}{2}w \cdot RTT_{\min} + \frac{1}{2}(w-W)^2 \cdot \frac{PS}{B}$	$\frac{1}{2}w \cdot RTT_{\min}$

PS: Packet size, B: Link bandwidth

TCPの理想挙動モデル (7)

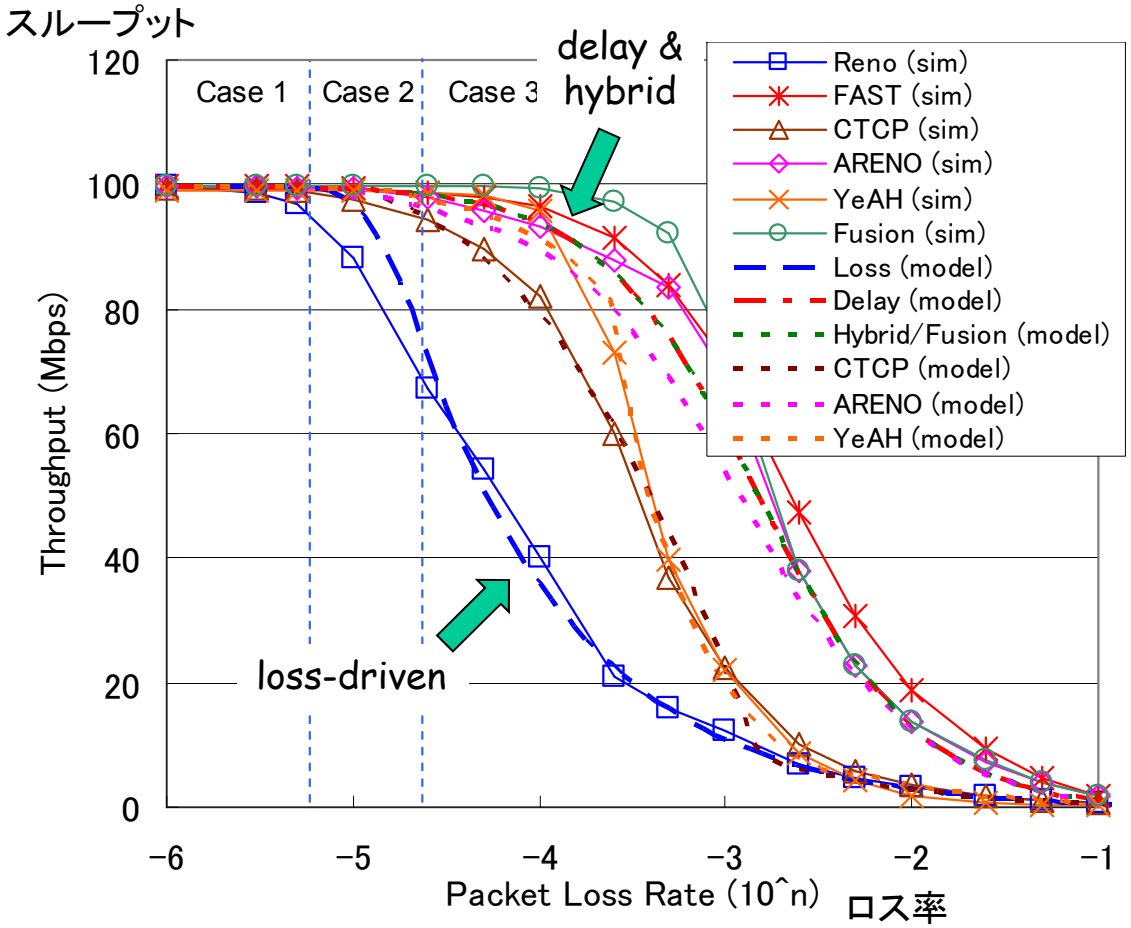
- 具体例の組込み (詳細略)

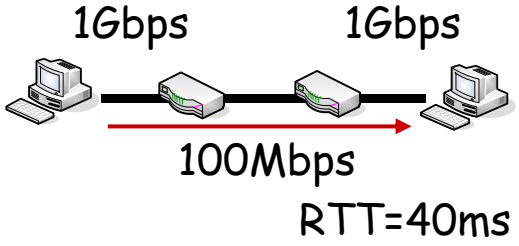
Hybrids	Window increase	Window decrease
Compound TCP	$0.125*cwnd^{0.75}$	1/2
ARENO	B/10Mbps	1/2～1
YeAH-TCP	Scalable TCP (1.01)	1/2, RTT_{min}/RTT, 7/8
TCP-Fusion	$B*D_{min}/(N*PS)$	RTT_{min}/RTT

D_{min}: timer resolution, N: # of flows

TCPの理想挙動モデル (8)

- モデル評価とシミュレーション評価





buffer size = BDP (constant)

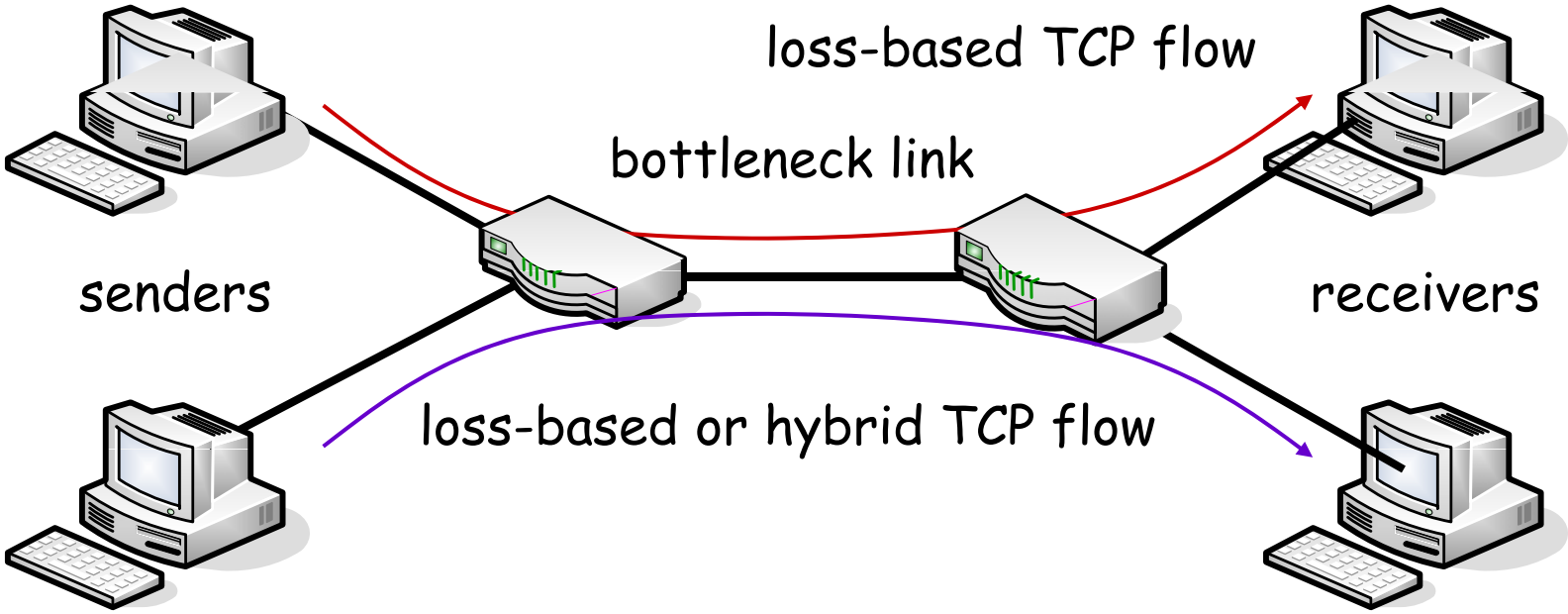
Packet loss rate : variable

ロス率が大きい時(w/2<W)、delay & hybrid のスループットは loss-driven よりもはるかに大きい(<u>効率性の改善</u>)。

Compound と YeAH の特性が落ちるのは、ウィンドウ減少率が大きいため。

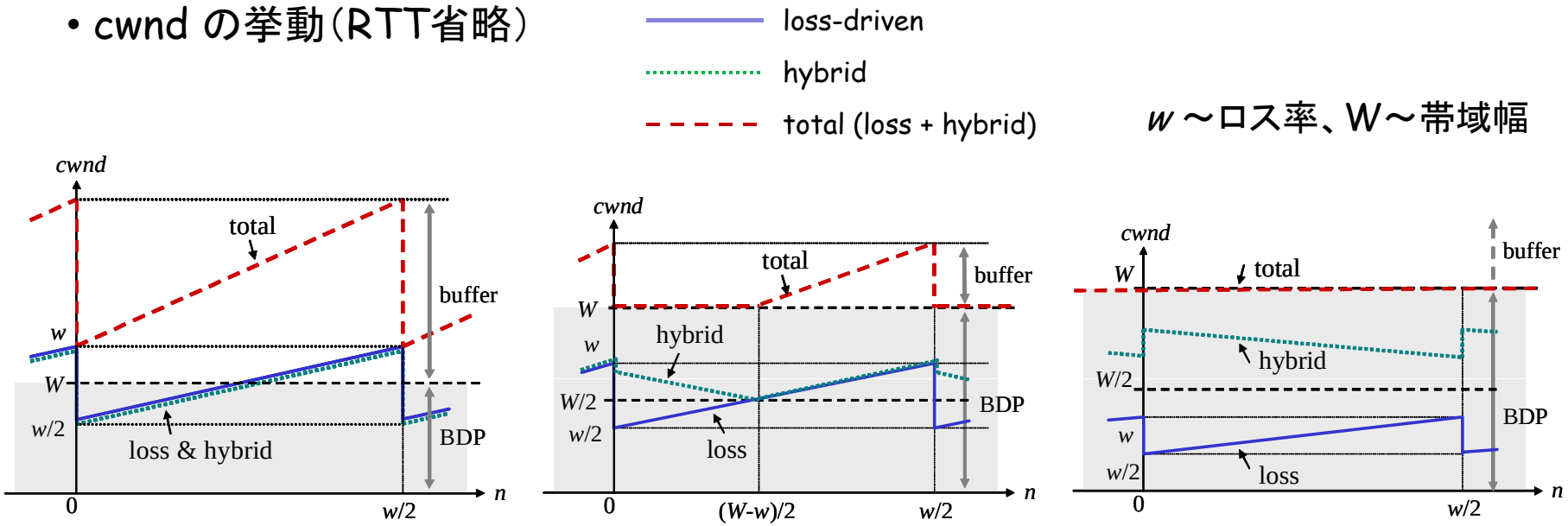
TCPの理想挙動モデル (9)

- 競合フローの場合



TCPの理想挙動モデル (10)

- cwnd の挙動(RTT省略)



(i) $W < w$ (low PLR)

always buffered
(loss mode)

バッファ大、ロス率小

(ii) $w < W < 2*w$ (medium PLR)

vacant → buffered
(delay → loss)

バッファ小、ロス率中

(iii) $2*w < W$ (high PLR)

always vacant
(delay mode)

ロス率大、常に空き帯域

TCPの理想挙動モデル (11)

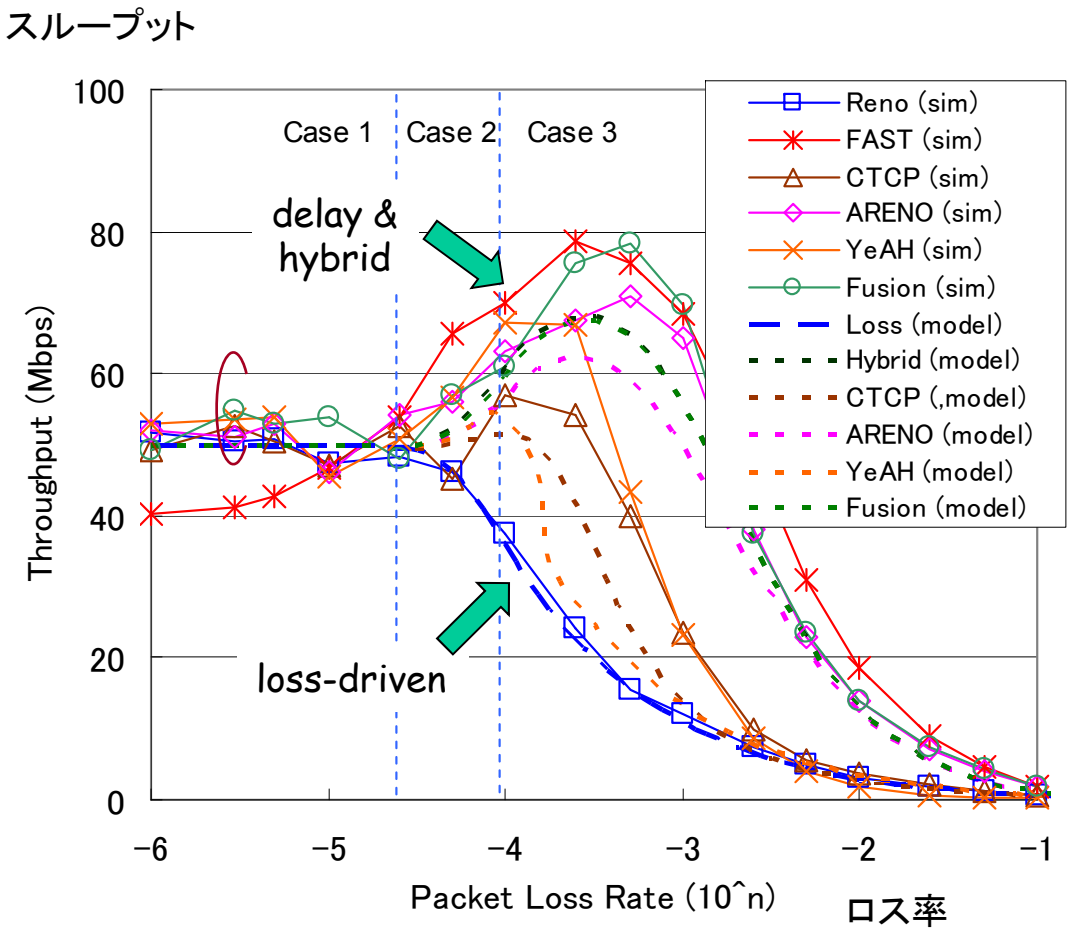
- 定式化

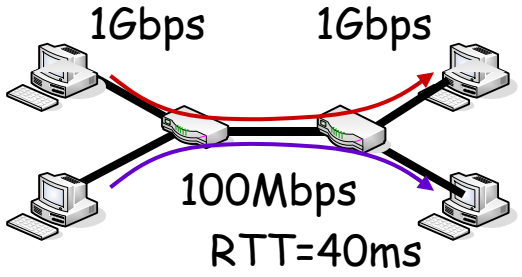
TCP	CA round	(i) $W < w$	(ii) $w \le W < 2w$	(iii) $2w \le W$
Loss	transmitted packets	$\frac{3}{8}w^2$	$\frac{3}{8}w^2$	$\frac{3}{8}w^2$
Hybrid	transmitted packets	$\frac{3}{8}w^2$	$\frac{3}{8}w^2 + \frac{1}{4}(W-w)^2$	$\frac{1}{2}w \cdot W - \frac{3}{8}w^2$
(common)	elapsed time	$\frac{1}{2}w \cdot RTT_{\min} + \frac{1}{4}w(3w-2W) \cdot \frac{PS}{B}$	$\frac{1}{2}w \cdot RTT_{\min} + \frac{1}{4}(2w-W)^2 \cdot \frac{PS}{B}$	$\frac{1}{2}w \cdot RTT_{\min}$

PS: Packet size, B: Link bandwidth

TCPの理想挙動モデル (12)

- モデル評価とシミュレーション評価





buffer size = BDP (constant)

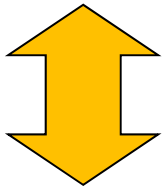
Packet loss rate : variable

ロス率が大きい時 (w<W)、delay & hybrid のスループットは loss-driven よりもはるかに大きい(<u>効率性の改善</u>)。

ロス率が小さい時 (w>W)、hybridはlossと同様に振る舞う(<u>親和性の改善</u>)。

TCPの理想挙動モデル (13)

- Hybrid TCPの利点
  - 空き帯域が存在する時（パケットロス率が大きい時）、スループット効率を大きく改善（delay-drivenの利点）
  - 空き帯域が存在しない時（バッファサイズが大きい時）レガシーTCPとの親和性を実現（loss-drivenの利点）



- Hybrid TCPの弱点
  - バッファサイズが大きい時、delay-drivenの利点が出ない（レガシーTCPはもう要らない？）

さて、どうしましょう。。。

ハイブリッドTCPのまとめ

Hybrid TCPのまとめ

- スループット効率と親和性の両立
  - リアルタイムストリーミング、大規模ファイルダウンロードへの適用
  - 無線環境でも有効？
  - CUBIC-TCPやCompound-TCPとの親和性
    - CUBIC-TCP: Linuxデフォルト
    - Compound-TCP: Windows
  - その他のメトリック
    - RTT公平性、Mice/Elephant、収束速度、etc…
  - ただし、あくまでも有効なのは delay-driven