画像情報特論 (2)
Advanced Image Information (2)

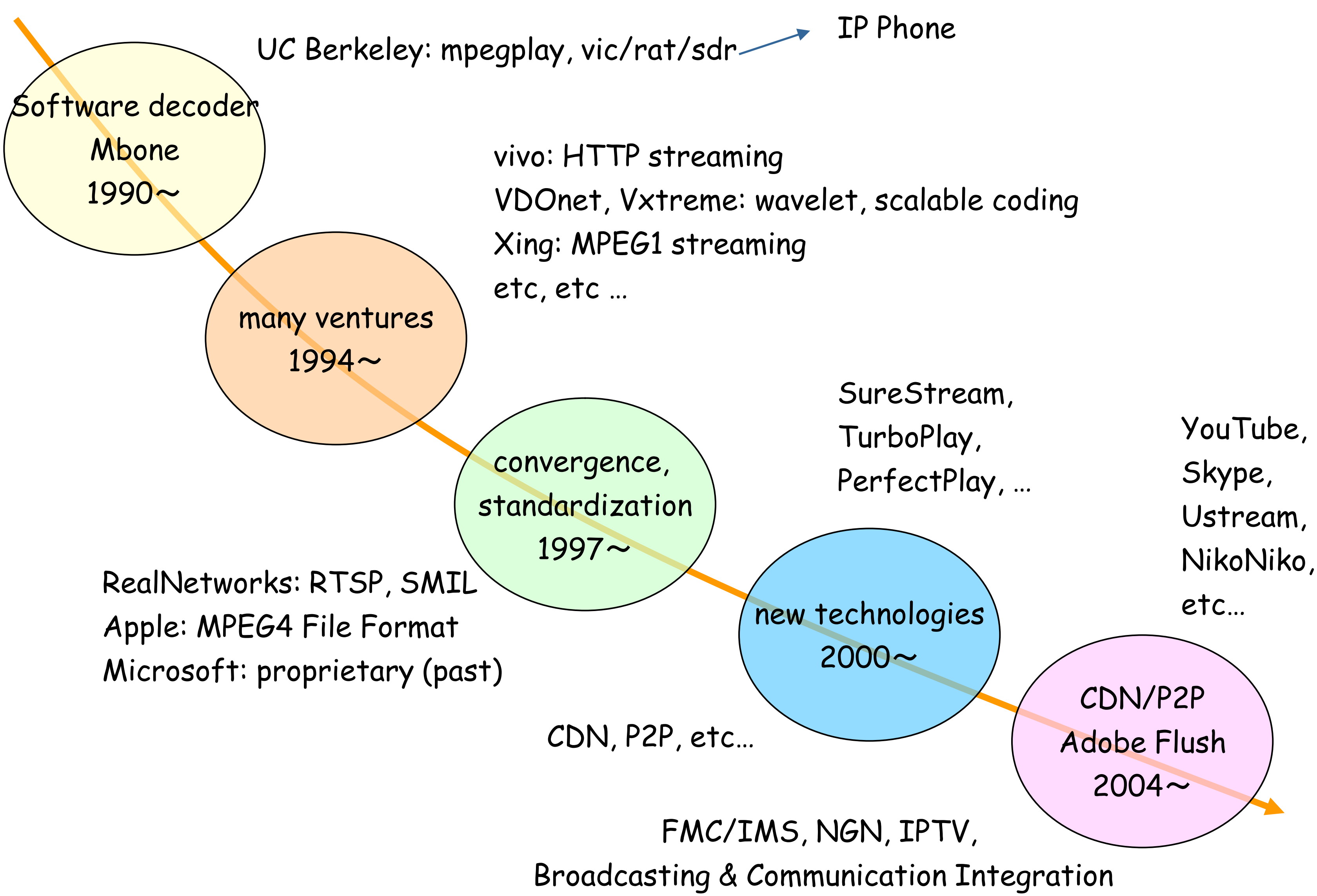
TCP/IP for Video Streaming

情報理工学専攻　甲藤二郎

E-Mail: katto@waseda.jp

Background

History of Streaming



Protocol Stack for Streaming

- Network Architecture

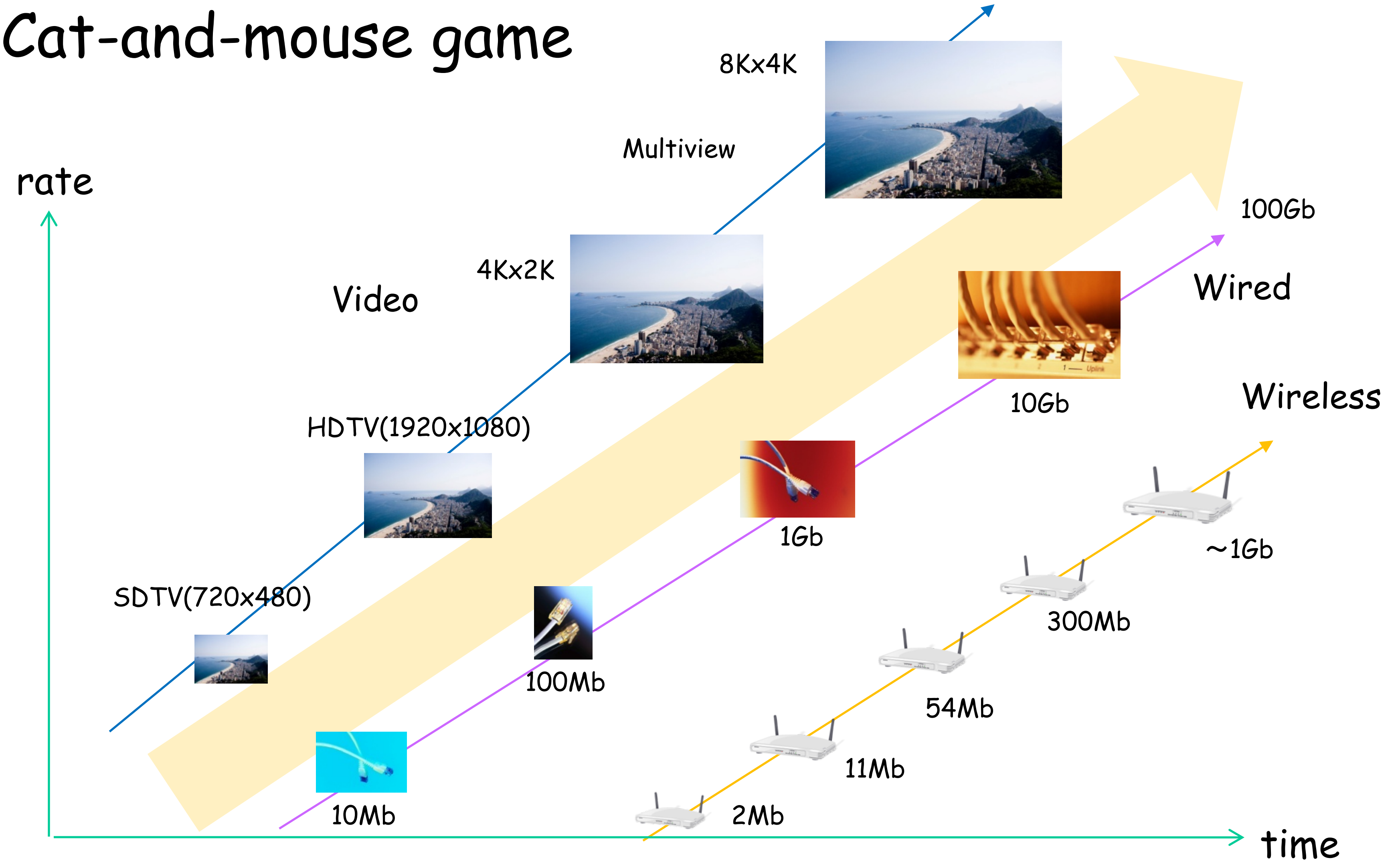
VoIP, IPTV and streaming shares almost common protocol stack

application (L7)	video (H.264 etc...)	audio	SDP	layout (HTML. SMIL)
adaptation	RTP / RTCP		RTSP, SIP, SAP*	HTTP
transport (L4)	UDP / TCP / DCCP		TCP / UDP / SCTP	
network (L3)	IP (IPv4, IPv6, IP-multicast)			
datalink & physical (L2 & L1)	actual networks (802.3 (ethernet), 802.11 (WiFi), etc)			

* SAP: delivered by IP-multicast for program advertisement

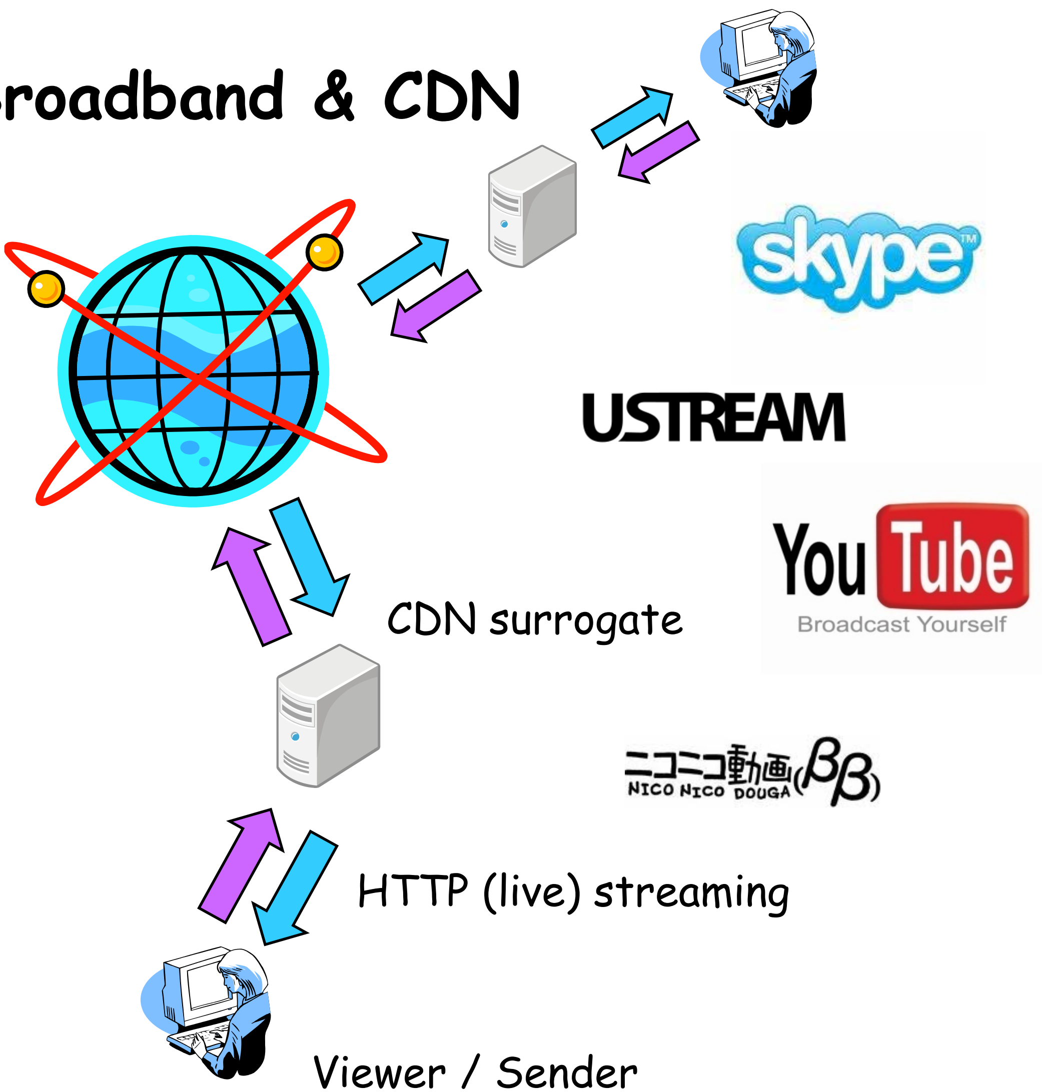
Networks and Multimedia

- Cat-and-mouse game



Wired Networks

Broadband & CDN



RTP/UDP & RTSP & TFRC

→ <u>HTTP/TCP streaming</u>

- Broadband
- CDN (Akamai, Lime Networks)
- Firewall (port 80)
- …

One-way (on-demand / live)

Bi-directional (interactive)

Wireless Networks

Wireless specific problems

- Wireless LAN: IEEE 802.11
- Cellular: 3G, LTE, 4G
- WiMAX: IEEE 802.16
- Home Networks: DLNA
- (Satellite)
- …

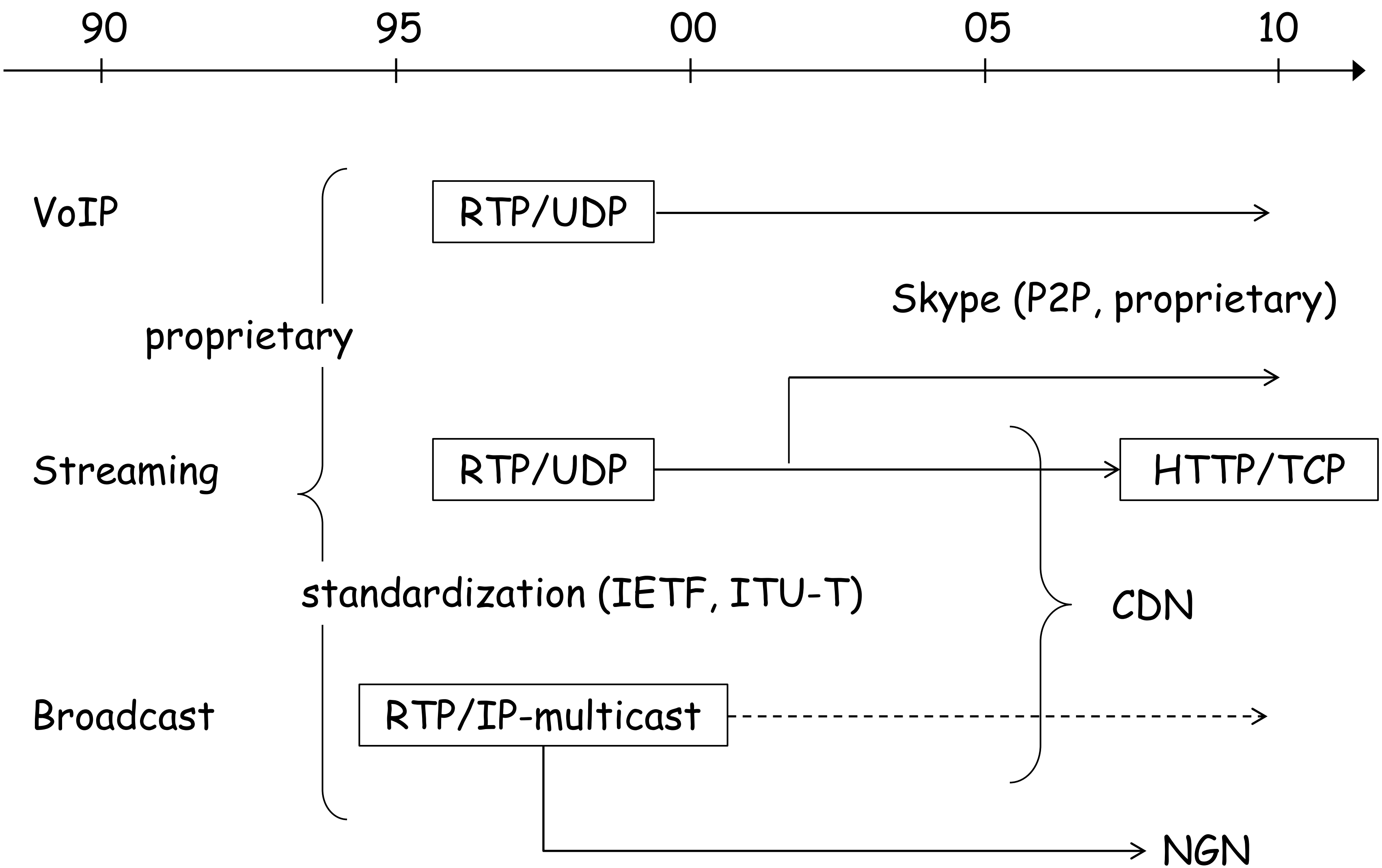
- **Wireless issues**

random errors, collisions, interference, delay increase

- **Multi-hop issues**

severe interference, lower throughput and higher delay

Protocol Transition

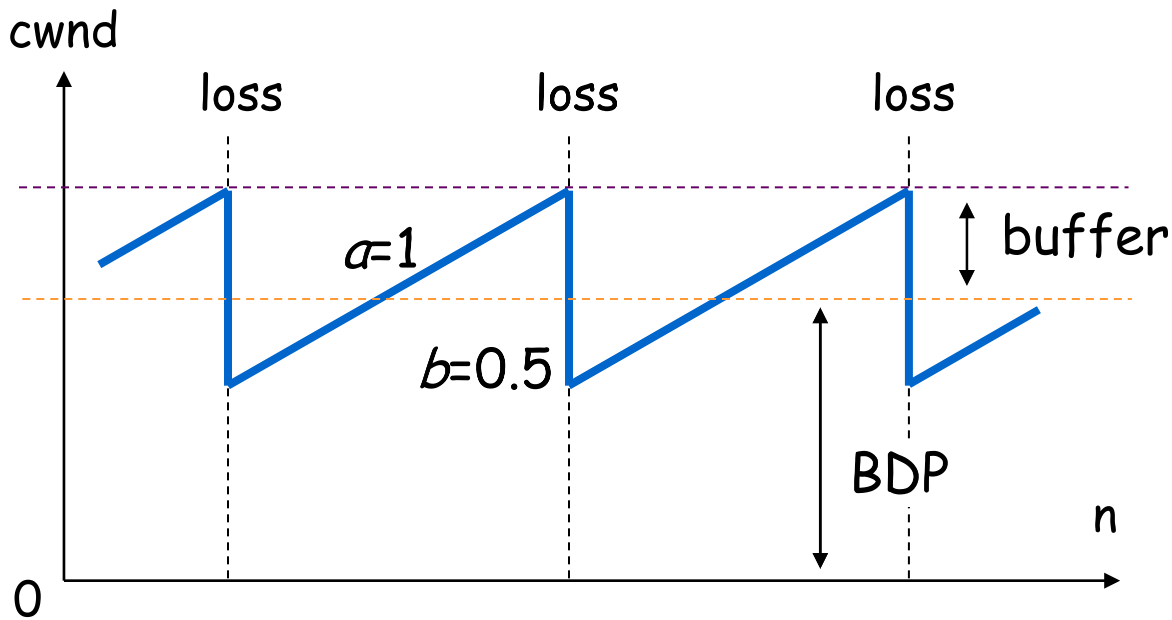


Overview

- Multimedia Delivery Techniques over IP Networks
- Broadband Wired Network
  - Transport techniques with efficiency, friendliness and low-delay properties
  - TCP and TFRC
- Wireless Network
  - Transport techniques towards low-delay transmission over wireless networks
  - Mobility
  - Multi-hop
  - Underwater sensor networks
- Network Simulators and Emulators

TCP Variants

TCP-Reno (loss-based)

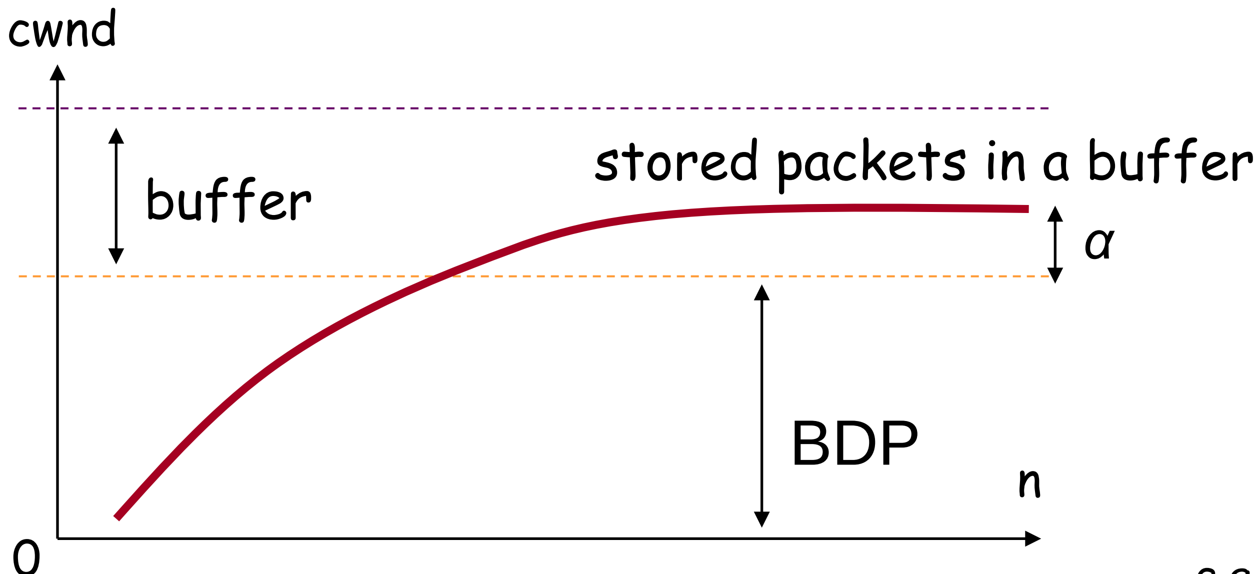


increase: cwnd = cwnd + 1/cwnd

decrease: cwnd = cwnd / 2

AIMD: additive increase multiplicative decrease

TCP-Vegas (delay-based)



e.g. α=1, β=3

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

stored packets in a buffer

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

decrease: $$cwnd = cwnd * 0.75$$

TCP problems ten years ago

- broadband wired networks
  - slow window increase (<u>inefficiency</u>)
- deployment of wireless networks
  - cannot distinguish wireless errors and buffer overflow

- TCP-Reno (NewReno, SACK) problem
  - Reno expels Vegas (<u>unfriendliness</u>)

TCP Variants in the 21th century

- **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) ... Linux
  - 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) ... Windows
  - Adaptive Reno (PFLDnet 2006)
  - YeAH-TCP (PFLDnet 2007)
  - TCP-Fusion (PFLDnet 2007)

Loss-based TCPs

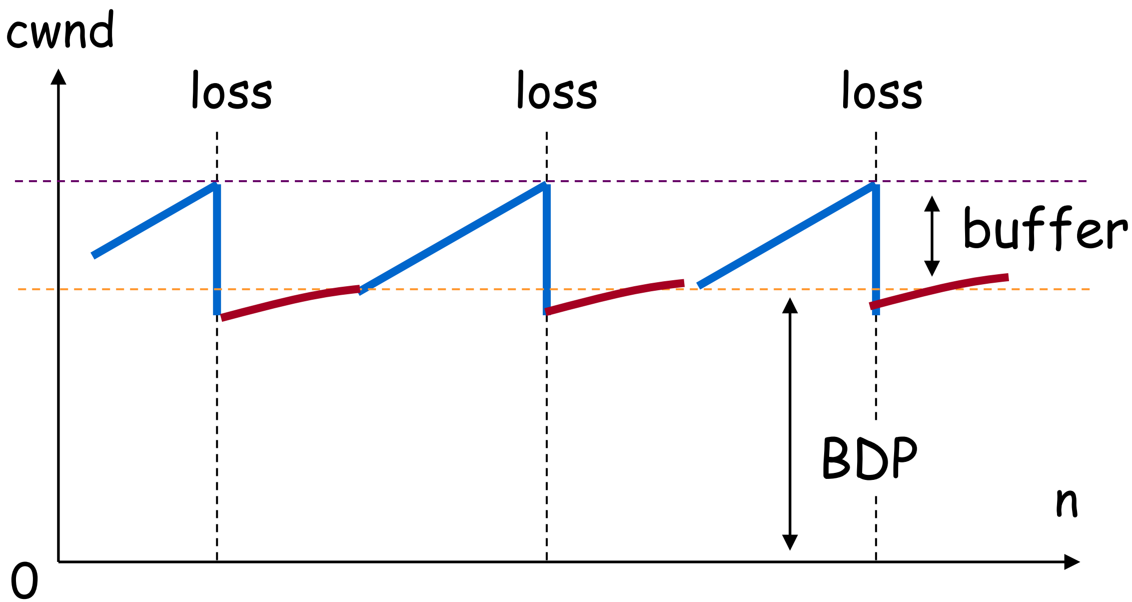
	Variants	Increase / Update (a)	Decrease (b)
	TCP-Reno	1	0.5
aggressive	HighSpeed TCP (HS-TCP)	$a(w)=\dfrac{2w^2\cdot b(w)\cdot p(w)}{2-b(w)}$ e.g. 70 (10Gbps, 100ms)	$b(w)=\dfrac{\log(w)-\log(W_{low})}{\log(W_{high})-\log(W_{low})}(b_{high}-0.5)+0.5$ e.g. 0.1 (10Gbps, 100ms)
aggressive	Scalable TCP (STCP)	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	$\alpha \leftarrow 2(1-\beta)\{1+10.5\cdot(t-TH)\}$	$\beta \leftarrow \begin{cases} 0.5 & for \left\lvert\dfrac{B(k+1)-B(k)}{B(k)}\right\rvert > 2 \\ \dfrac{RTT_{\min}}{RTT_{\max}} & otherwise \end{cases}$
adaptive	TCP-Westwood (TCPW)	1	$\begin{cases} RE * RTT_{\min}/PS & (not\ congested) \\ BE * RTT_{\min}/PS & (congested) \end{cases}$

Delay-based TCPs

a b

Variants	Update	Decrease
TCP-Vegas	$w \leftarrow \begin{cases} w+1 & (no\ congestion) \\ w & (stable) \\ w-1 & (early\ congestion) \end{cases}$	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 increase: loss mode ⇒ improvement of friendliness
- no RTT increase: delay mode ⇒ improvement of efficiency

Hybrid TCPs

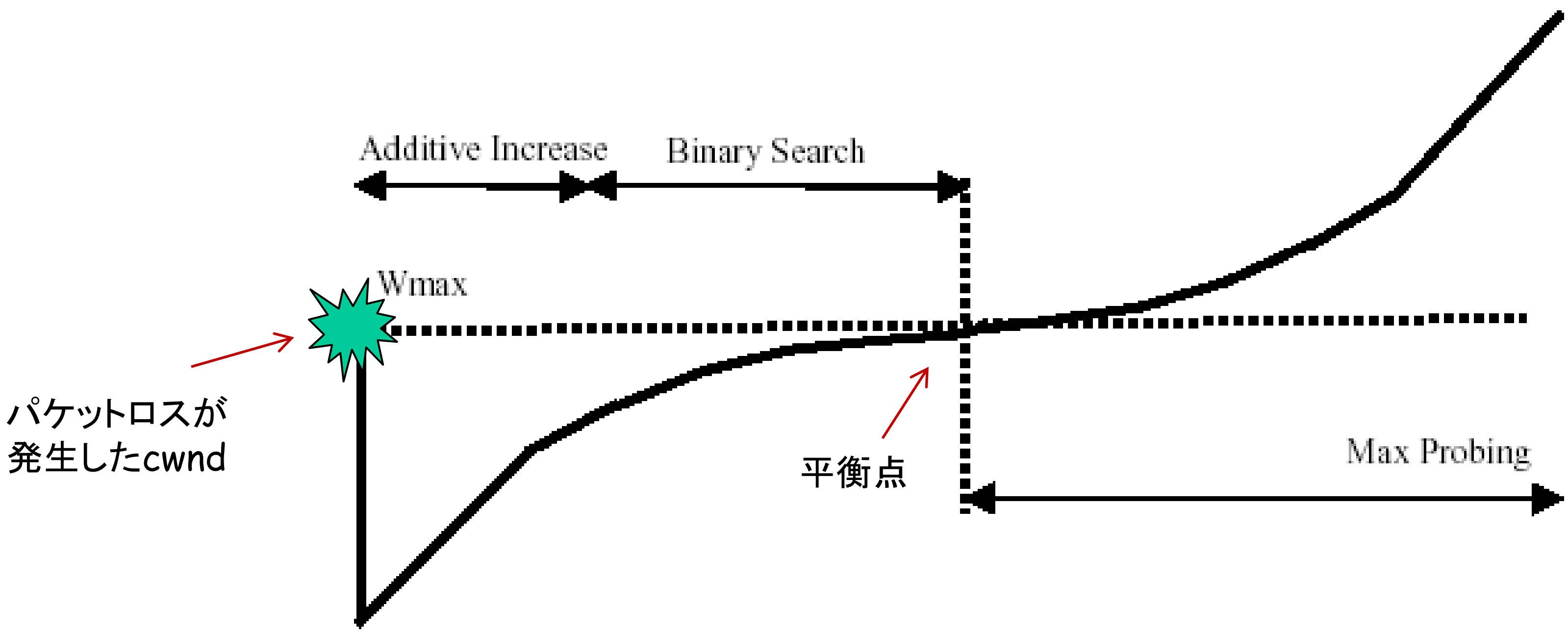
a b

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

CUBIC-TCP

BIC-TCP (1)

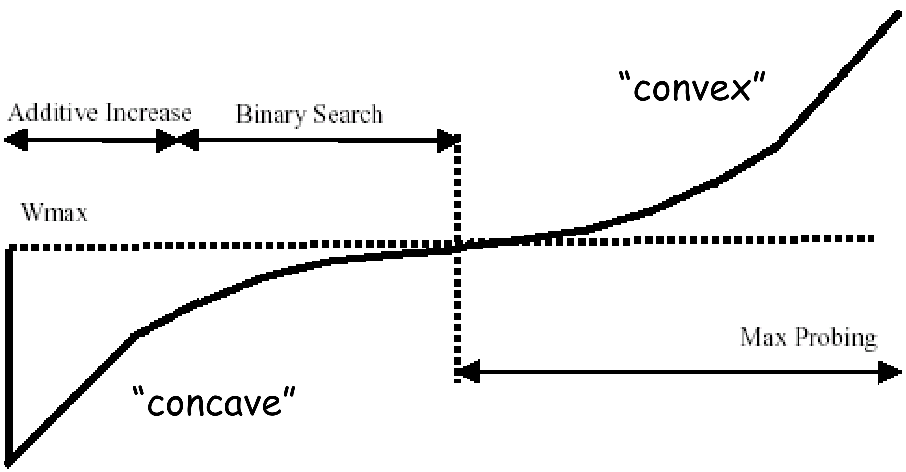
- Binary Increase Congestion Control



L.Xu et al: "Binary Increase Congestion Control (BIC) for Fast Long-Distance Networks," IEEE INFOCOM 2004.

BIC-TCP (2)

- Window Increase



binary search

```
if (cwnd < Wmax )
    Winc = (Wmax – cwnd) / 2;
else
    Winc = (cwnd - Wmax) / 2;

if (Winc > Smax)
    Winc = Smax;
elseif (Winc < Smin)
    Winc = Smin;

cwnd = cwnd + Winc / cwnd;
```

additive increase
(linear increase)

Wmax: cwnd when a last loss happened

Smax: maximum increase rate (e.g. 32)

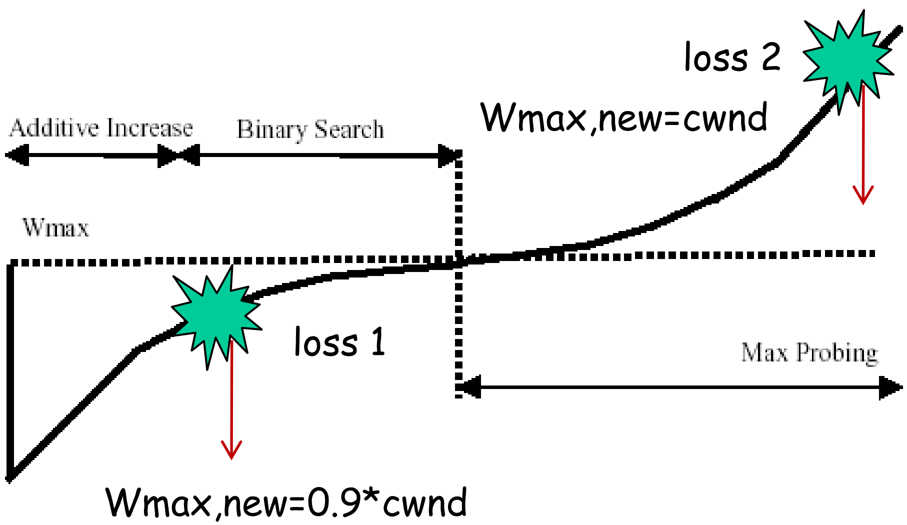
Smin: minimum increase rate (e.g. 0.01)

L.Xu et al: "Binary Increase Congestion Control (BIC) for Fast Long-Distance Networks," IEEE INFOCOM 2004.

BIC-TCP (3)

- Window Decrease

Wmax=target cwnd update



if (cwnd < Wmax)
    Wmax,new = cwnd * (2-β) / 2;
else
    Wmax,new = cwnd;

cwnd = cwnd * (1- β);

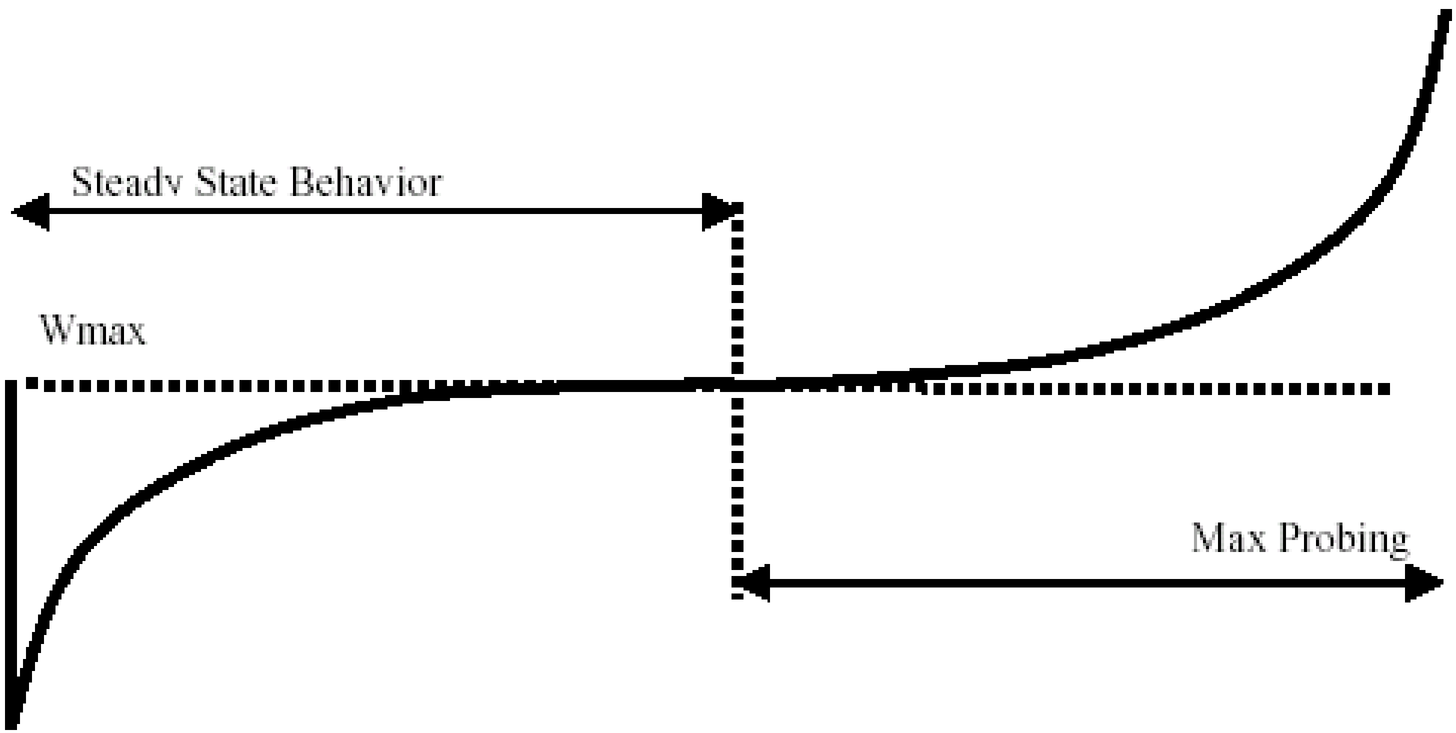
β: decrease rate (e.g. 0.2)

*0.9: give bandwidth to newly-coming flows
... "Fast Convergence"

L.Xu et al: "Binary Increase Congestion Control (BIC) for Fast Long-Distance Networks," IEEE INFOCOM 2004.

CUBIC-TCP (1)

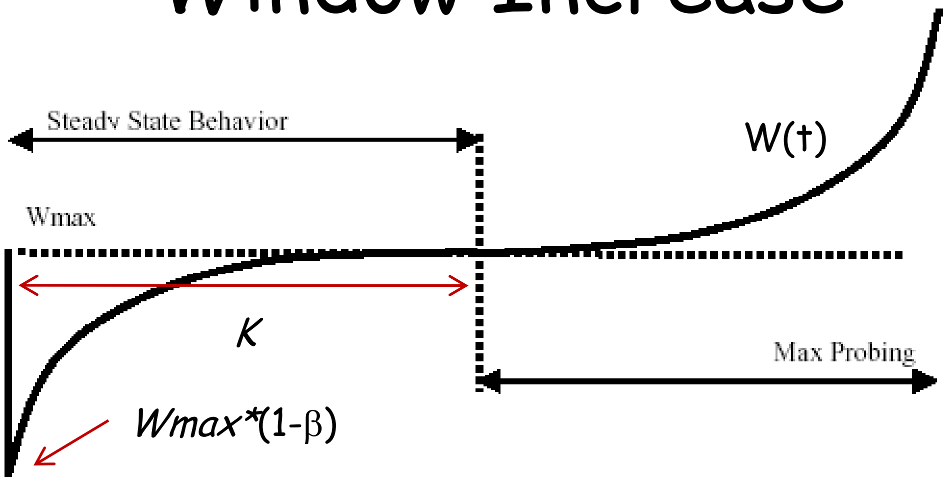
- Cubic approximation of BIC-TCP



S.Ha et al: "CUBIC: A New TCP Friendly HighSpeed TCP Variant", ACM SIGOPS Review, 2008.

CUBIC-TCP (2)

- Window Increase



$$W(t) = C * (t - K)^3 + W_{\max}$$

$$K = \sqrt[3]{\frac{W_{\max}\beta}{C}}$$

equivalent to Reno

$$W_{tcp}(t) = W_{\max}(1-\beta) + 3\frac{\beta}{2-\beta}\frac{t}{RTT}$$

```
/* cubic function */
Winc = W(t+RTT) – cwnd;

cwnd = cwnd + Winc / cwnd;

/* TCP mode */
if ( Wtcp > cwnd )
   cwnd = Wtcp;
```

※ window decrease is the same as BIC

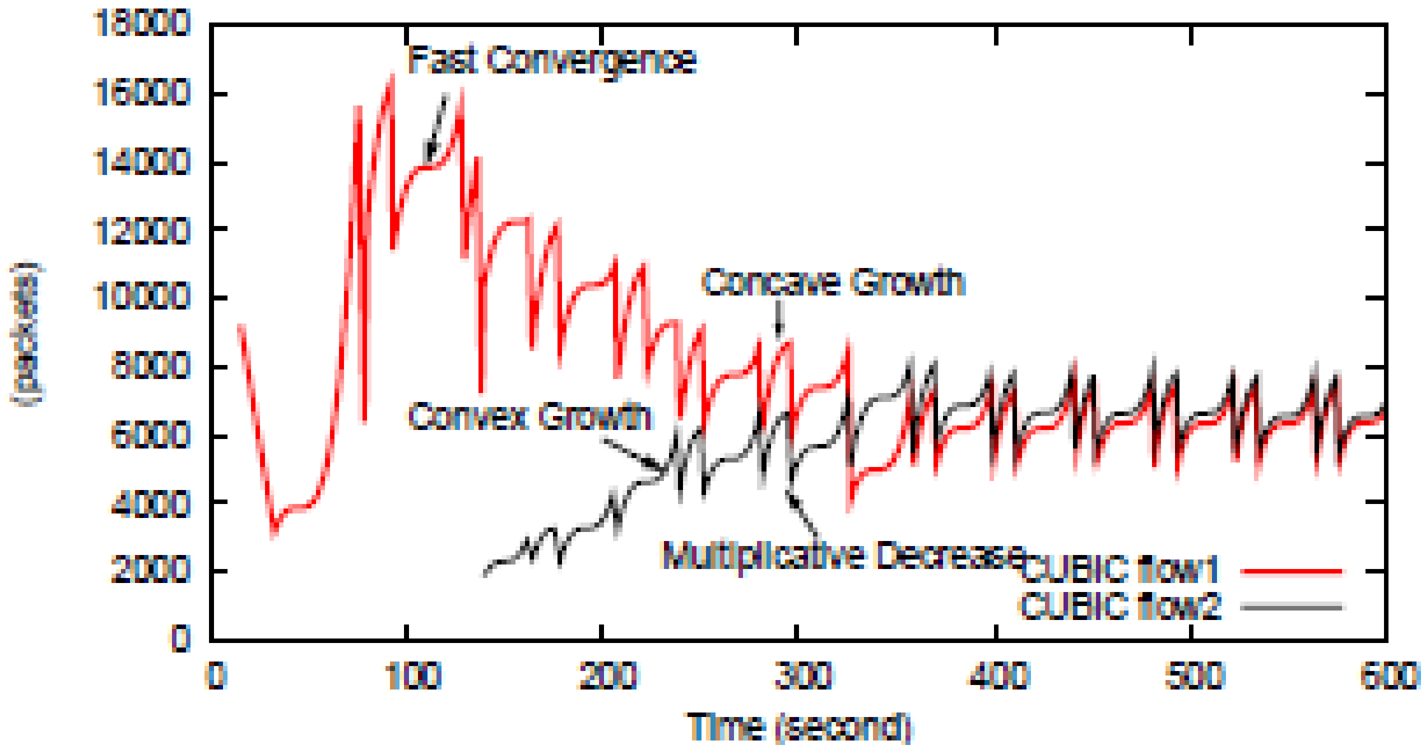
β: decrease rate (e.g. 0.2)

C: constant (e.g. 0.4)

S.Ha et al: "CUBIC: A New TCP Friendly HighSpeed TCP Variant", ACM SIGOPS Review, 2008.

CUBIC-TCP (3)

- CUBIC's cwnd behavior



S.Ha et al: "CUBIC: A New TCP Friendly HighSpeed TCP Variant", ACM SIGOPS Review, 2008.

CUBIC-TCP (4)

- Advantages
  - stability
  - "intra-protocol fairness" among multiple CUBIC flows
- Disadvantages
  - heavy buffer occupancy and delay increase ($\Leftrightarrow$ delay-based)
  - "inter-protocol unfairness" against other TCP flows
    - "Linux beats Windows!" (vs. Compound TCP)

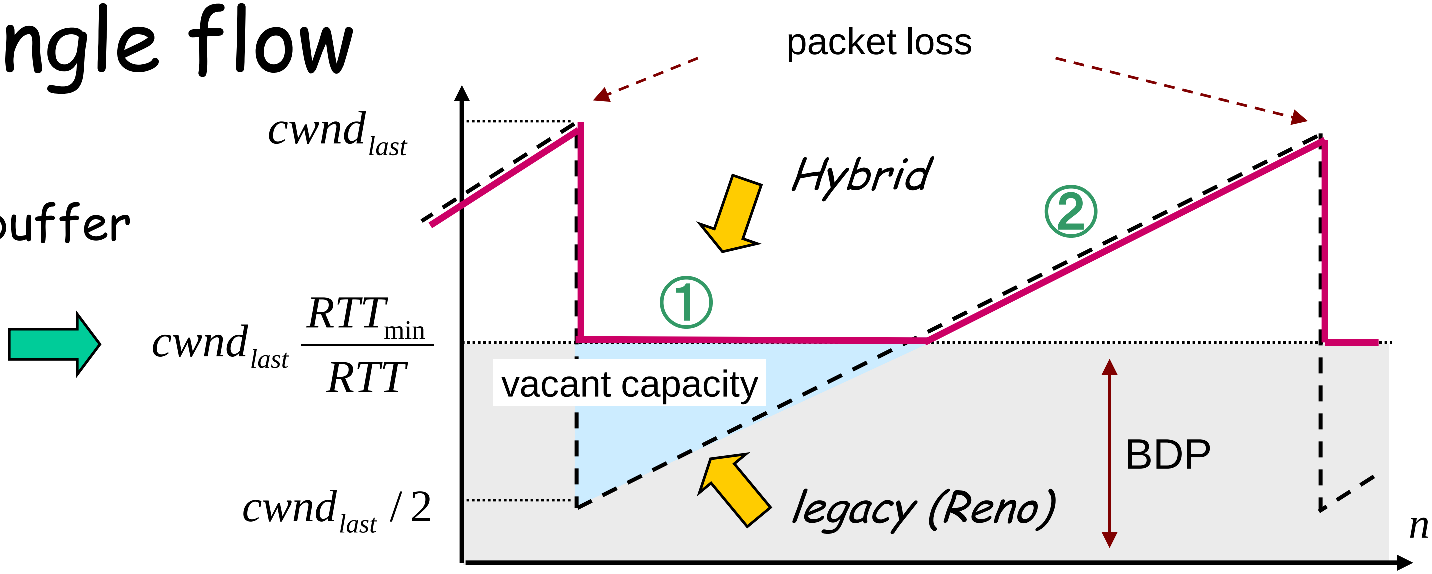
K.Munir et al: "Linux beats Windows! or the Worrying Evolution of TCP...", PFLDNet 2007.

Hybrid TCPs (and its Performance Analysis)

Hybrid TCP (1)

- single flow

clearing buffer
(TCPW)

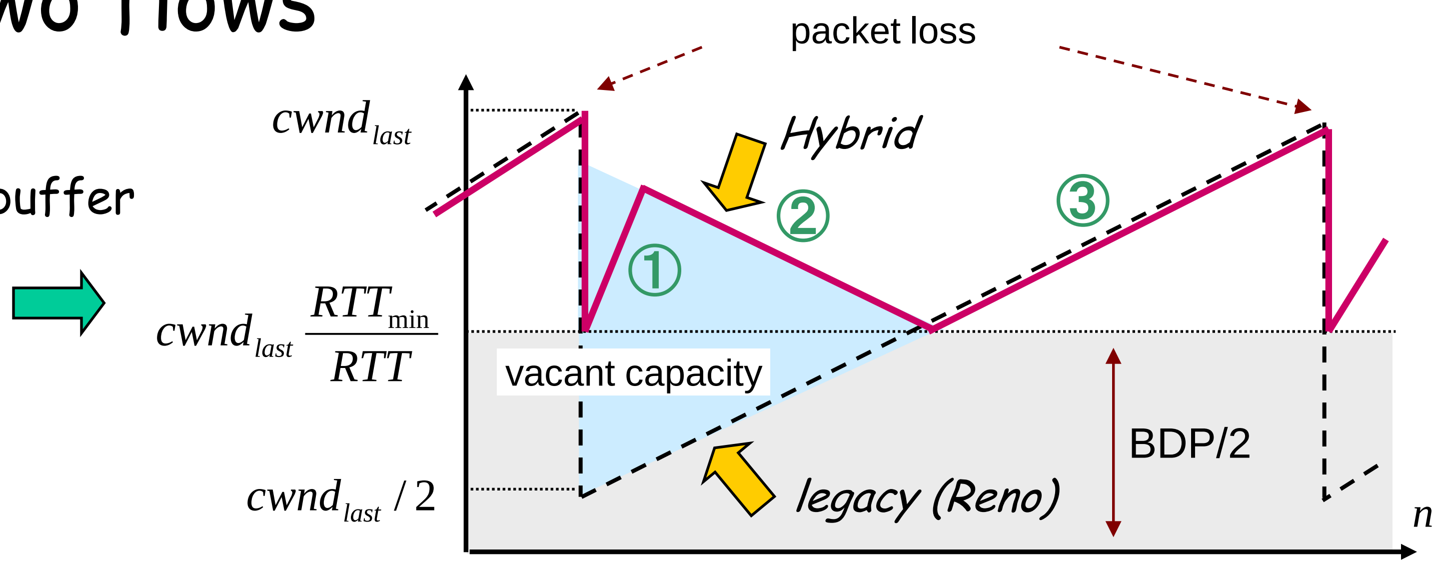


adaptive switching of two modes (loss & delay):
① constant rate until RTT increases (delay mode) : "efficiency" and "low delay"
② performs as Reno when RTT increases (loss mode) : "friendliness"

Hybrid TCP (2)

- two flows

clearing buffer
(TCPW)



adaptive switching of two modes (loss & delay):

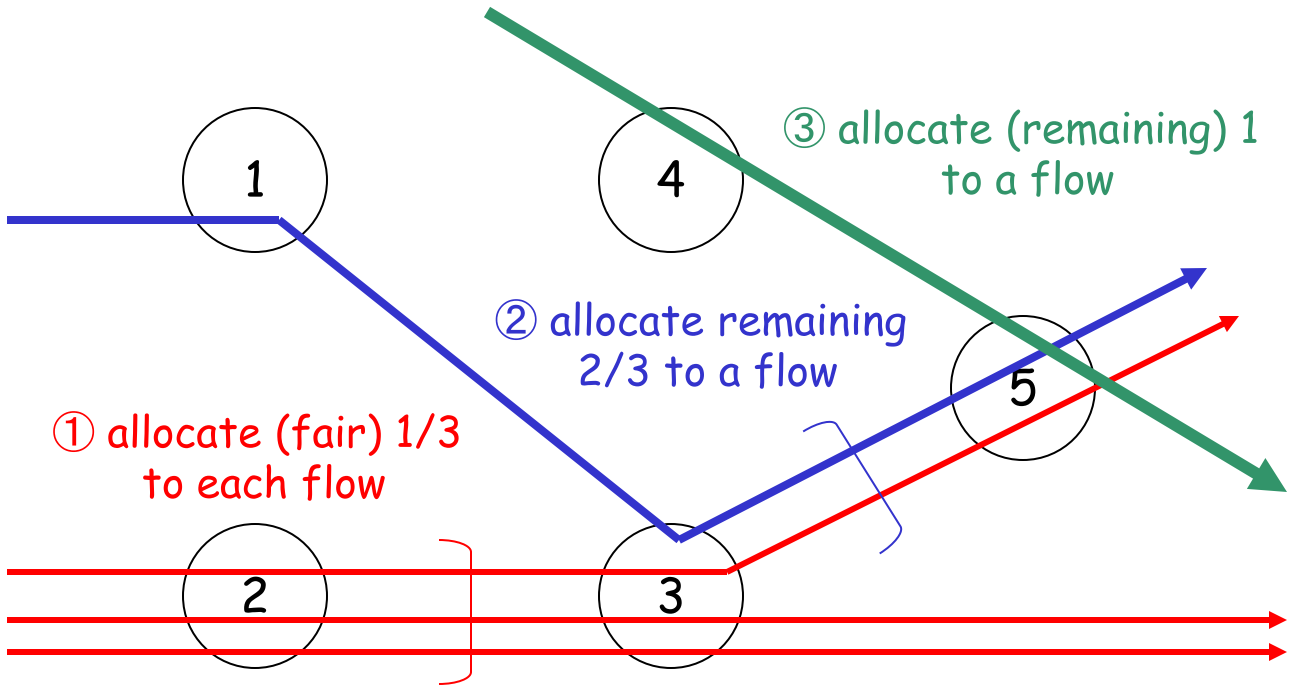
① fast cwnd increase (delay … "efficiency")

② mild cwnd decrease (delay … congestion avoidance)

③ performs as Reno when RTT increases (loss … "friendliness")

Min-Max Fair (ideal case)

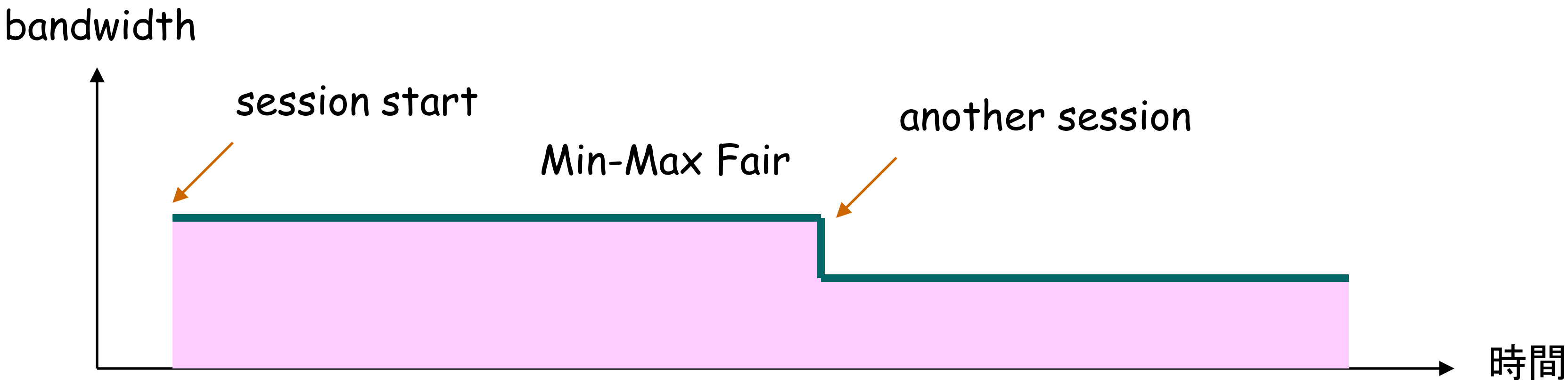
- Min-Max-Fair: allocate “maximum bandwidth” to a user who has “minimum bandwidth”



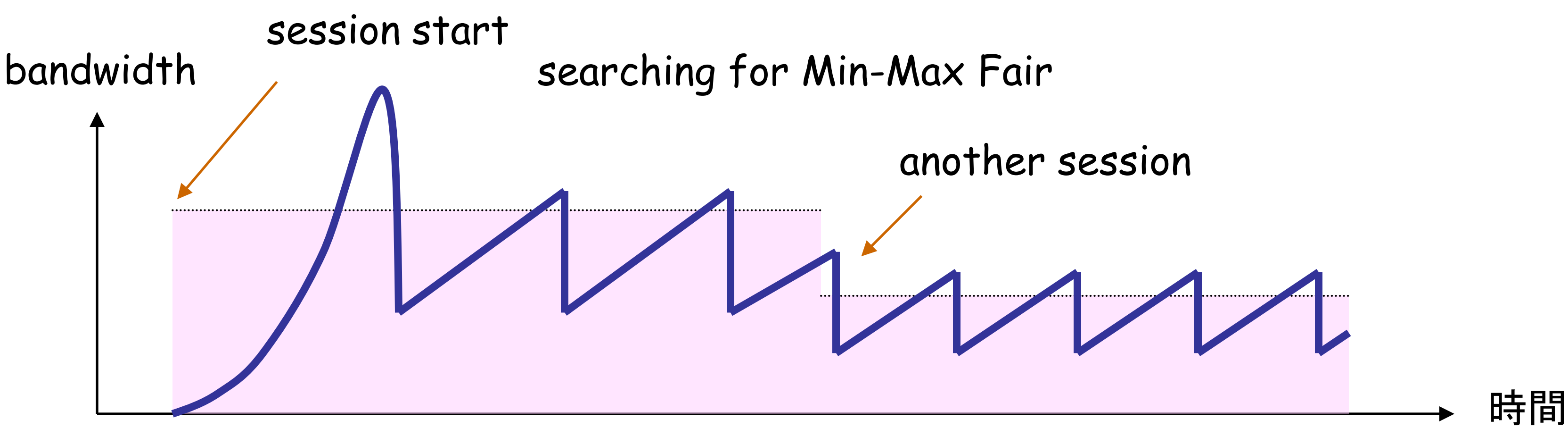
D.Bertsekas and R.Gallager: “Data Networks,” Prentice Hall.

TCP's objective

Ideal:



TCP Reno



TCP behavior model (1)

- model definition
  - Loss-mode (TCP-Reno) :
    - cwnd += 1 (per “RTT round”)
    - cwnd *= 1/2 (when a packet loss is detected)
  - Delay-mode :
    - fill a “pipe” (fully utilize a link) without causing RTT increase
  - Hybrid :
    - works in delay mode when RTT is not increased
    - works in loss mode when RTT is increases (i.e. when packets are buffered)
    - mode selection: cwnd = max($cwnd_{loss}$, $cwnd_{delay}$)

TCP behavior model (2)

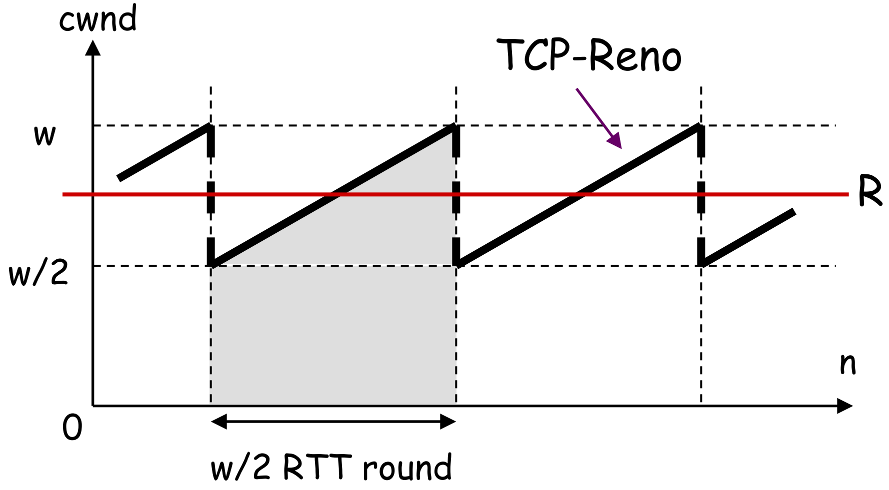
- parameter definition
  - w : cwnd when a packet loss is detected
  - W : cwnd which just fills a pipe ~ BDP
  - p : packet loss rate

- assumption
  - packet loss due to buffer overflow is equivalent to packet loss due to random error

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

TCP behavior model (3)

- TCP friendly model



w: cwnd when a packet loss is detected
p: packet loss rate
RTT: round trip time

R: TCP equivalent rate

# of transmitted packets until a packet loss is detected

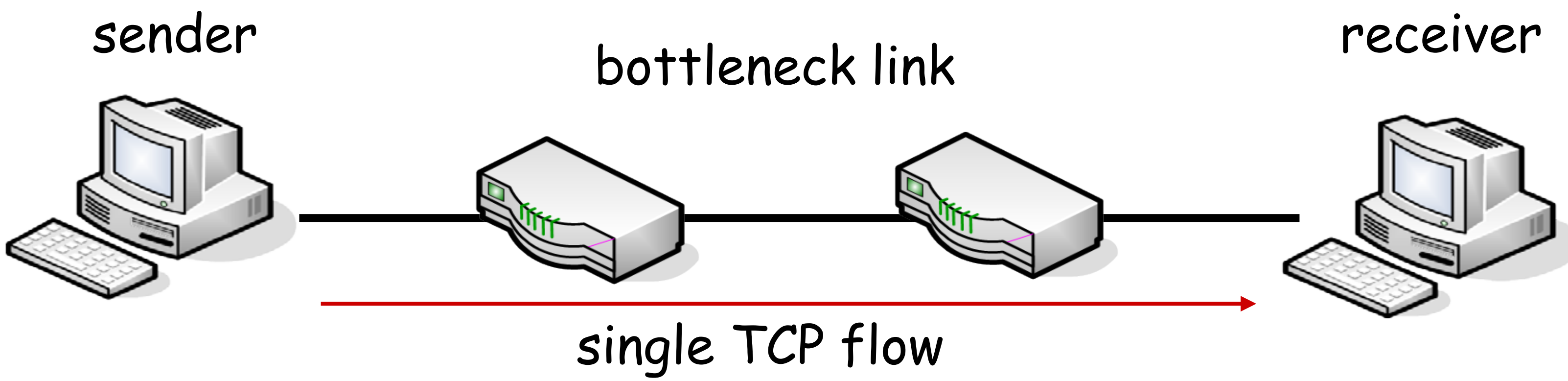
= area of a trapezoid

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

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

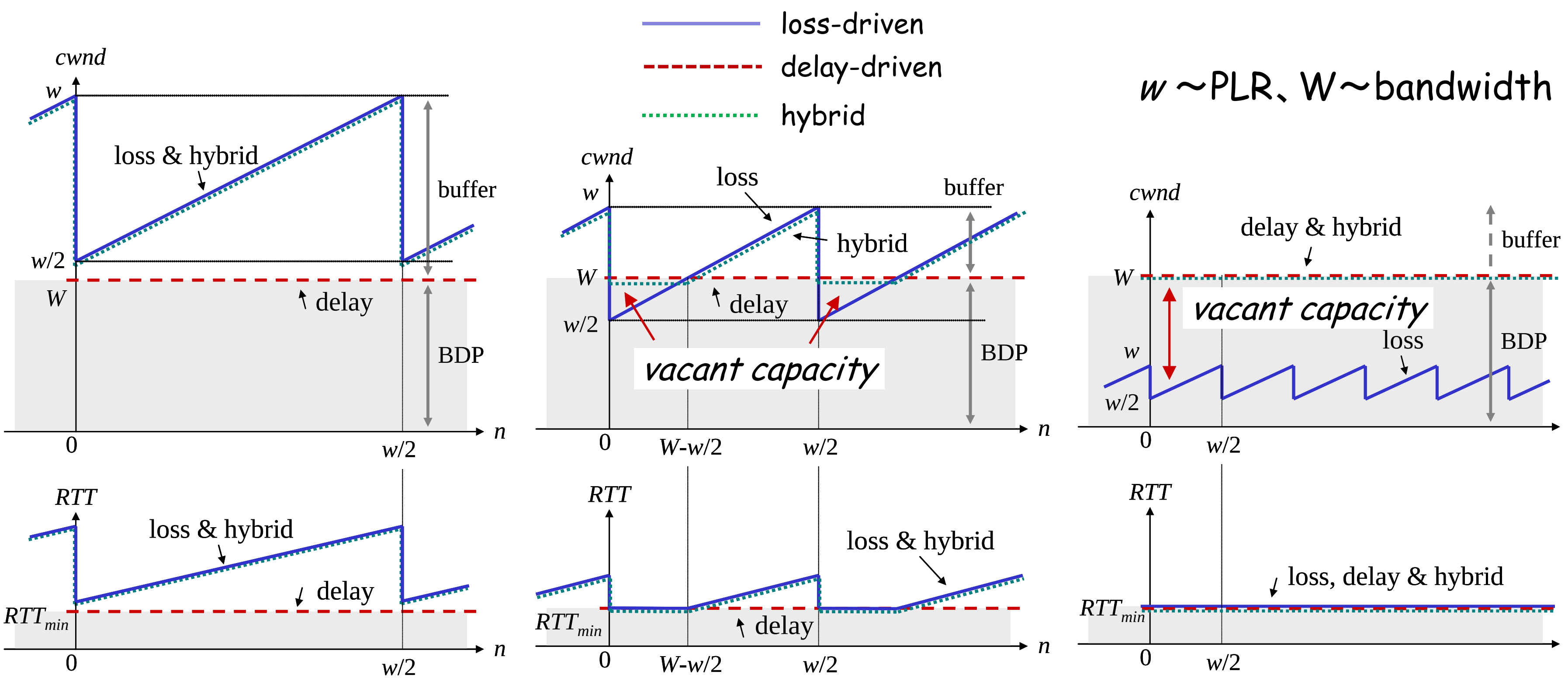
TCP behavior model (4)

- single flow



TCP behavior model (5)

- cwnd & RTT behaviors of ideal models (single flow case)



(i) $W < w/2$

large buffer, small PLR
(always loss-mode)

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

small buffer, medium PLR
(delay ↔ loss)

(iii) $w < W$

large PLR, always vacant
(always delay-mode)

TCP behavior model (6)

- formulation

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 behavior model (7)

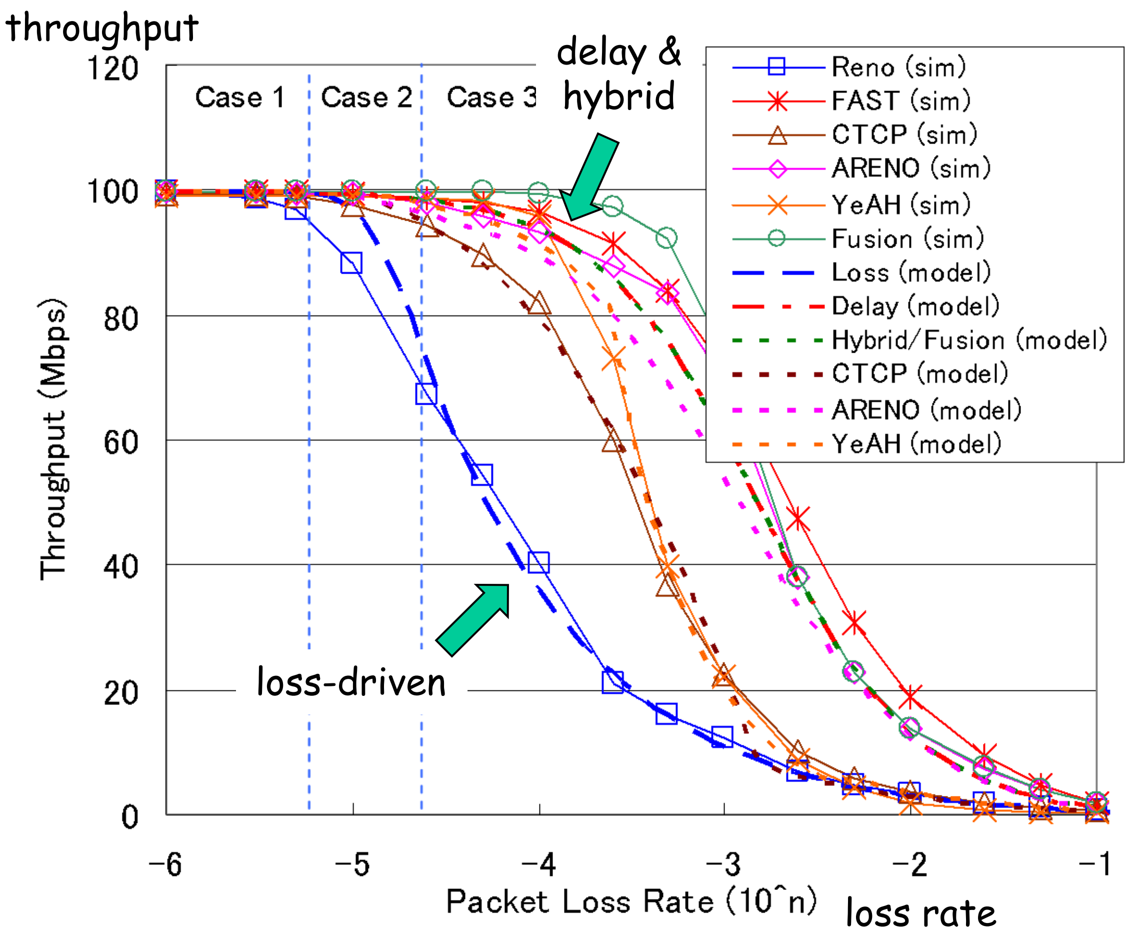
- abstraction of actual hybrids

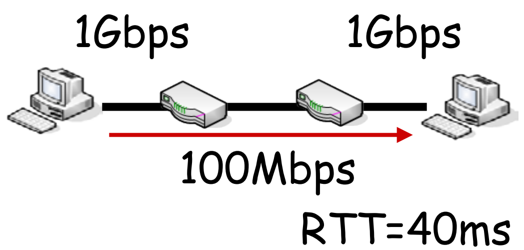
Hybrids	Window increase	Window decrease
Compound TCP	$0.125 * cwnd^{0.75}$	1/2
ARENO	B/10Mbps	1/2～1
YeAH-TCP	Scalable TCP (1.01)	1/2, RTT_{min}/RTT, 7/8
TCP-Fusion	$B * D_{min}/(N * PS)$	RTT_{min}/RTT

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

TCP behavior model (8)

- evaluation by models and simulations





buffer size = BDP (constant)

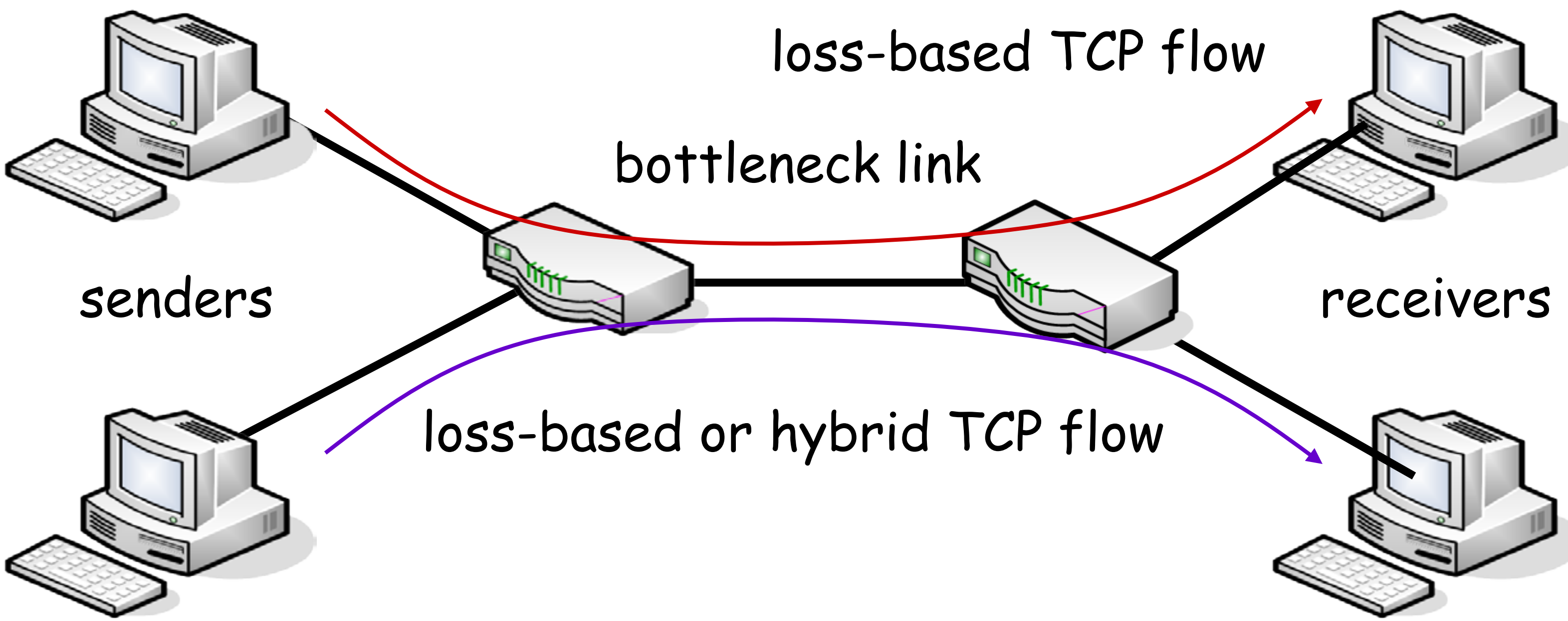
Packet loss rate : variable

when PLR is large (w/2<W), throughputs of delay & hybrid are much larger than that of loss-mode (i.e. <u>efficiency</u>)

degradation of Compound & YeAH is due to fixed window decrease

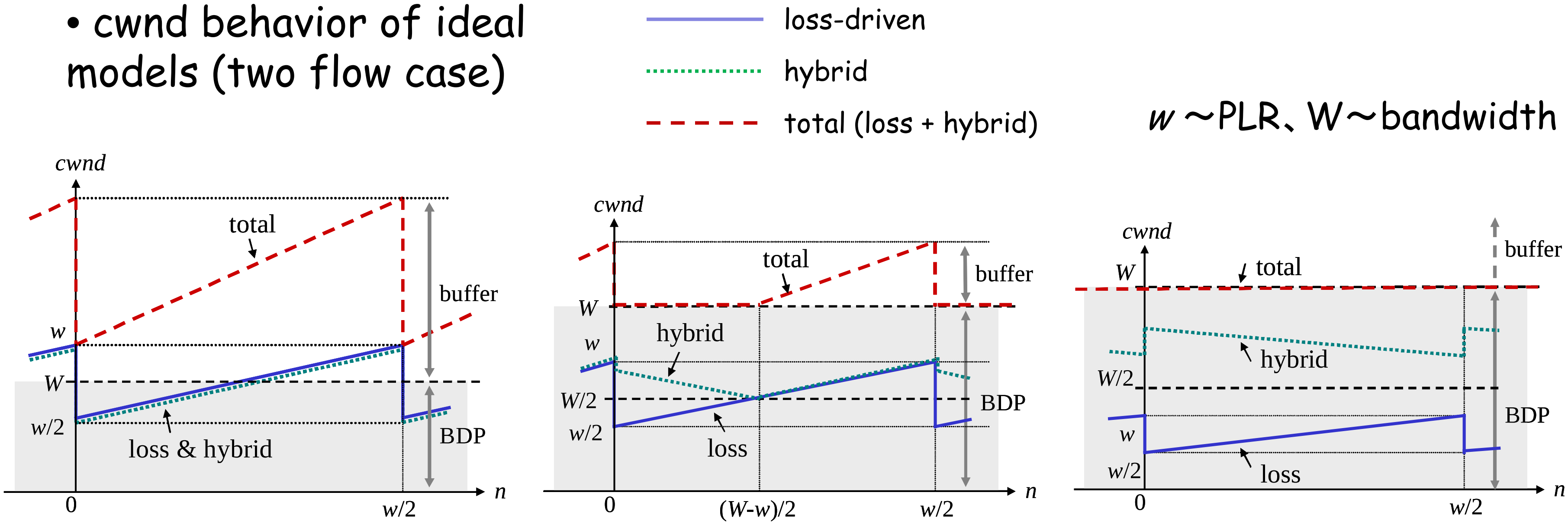
TCP behavior model (9)

- two flows (competing)



TCP behavior model (10)

- cwnd behavior of ideal models (two flow case)



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

always buffered
(loss mode)

large buffer, small PLR

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

vacant → buffered
(delay → loss)

small buffer, medium PLR

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

always vacant
(delay mode)

large PLR, always vacant

TCP behavior model (11)

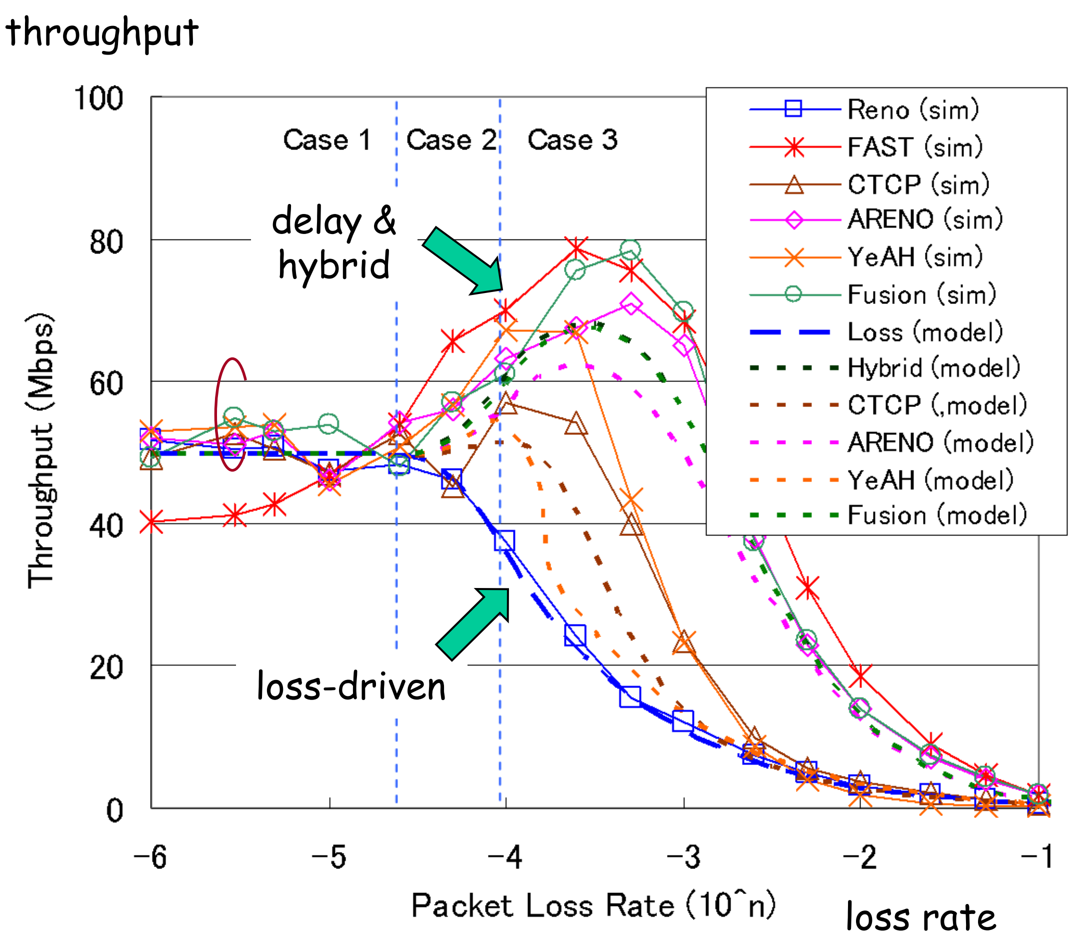
- formulation

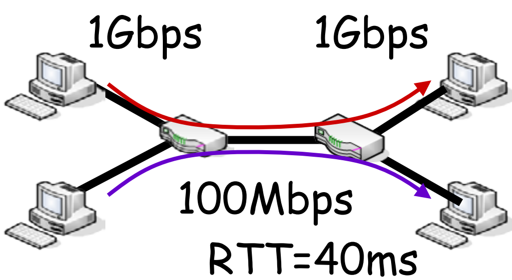
TCP	CA round	(i) $W < w$	(ii) $w \leq W < 2w$	(iii) $2w \leq W$
Loss	transmitted packets	$\frac{3}{8}w^2$	$\frac{3}{8}w^2$	$\frac{3}{8}w^2$
Hybrid	transmitted packets	$\frac{3}{8}w^2$	$\frac{3}{8}w^2 + \frac{1}{4}(W-w)^2$	$\frac{1}{2}w \cdot W - \frac{3}{8}w^2$
(common)	elapsed time	$\frac{1}{2}w \cdot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

PS: Packet size, B: Link bandwidth

TCP behavior model (12)

- evaluation by models and simulations





buffer size = BDP (constant)

Packet loss rate : variable

when PLR is large (w<W), throughputs of delay & hybrid are much larger than that of loss-mode (**efficiency**)

when PLR is low (w>W), hybrid behaves similar to loss-mode (**friendliness**)

TCP behavior model (13)

- Advantages of Hybrid TCP
  - when vacant capacity exists (or PLR is large), throughput efficiency is greatly improved (advantage of delay-mode)
  - when no vacant capacity exists (or buffer size is large), friendliness to legacy TCP (i.e. Reno) is achieved (advantage of loss-mode)

- Disadvantages of Hybrid TCP
  - when buffer size is large, delay-mode is never activated …

Summary of Hybrid TCP

Hybrid TCP

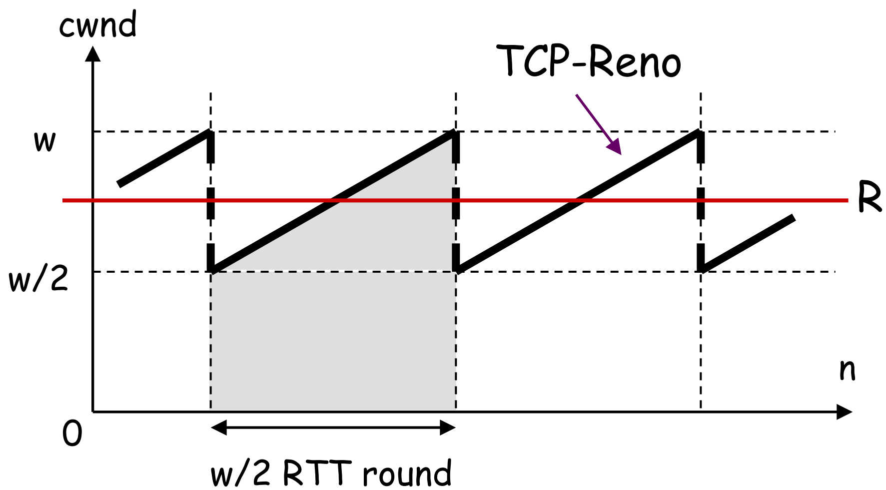
- "Efficiency", "Friendliness" and "Low delay"
  - can be applied to real-time streaming and large file download
  - might be effective in wireless networks
  - friendliness to CUBIC-TCP or Compound-TCP
    - CUBIC-TCP: Linux default
    - Compound-TCP: Windows
  - other metrics
    - RTT fairness, mice/elephant (short-lived or long-lived), convergence speed, etc…

  - efficiency is brought by delay-mode

TCP Equations

TCP Modeling

- TCP-Reno Equivalent Rate



w: cwnd when packet is lost

p: PLR

RTT: round trip time

R: equivalent rate

b: # of delayed ACKs

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

with timeout consideration

$$\Rightarrow R_{loss} = \frac{PS}{RTT\sqrt{\dfrac{2bp}{3}} + t_{RTO,loss} \cdot 3\sqrt{\dfrac{3bp}{8}} \cdot p(1+32p^2)}$$

J.Padhye et al: "Modeling TCP Throughput: A Simple Model and its Empirical Validation", ACM SIGCOMM 1998.

TCP Westwood

FSE: Fair Share Estimates

- Duplicate ACKs

$$ssthresh = FSE * RTT_{\min}$$
$$if\ (cwnd > ssthreh)\ cwnd = ssthresh$$

- Timeout

in TCP-Reno case

$$ssthresh = cwnd / 2$$

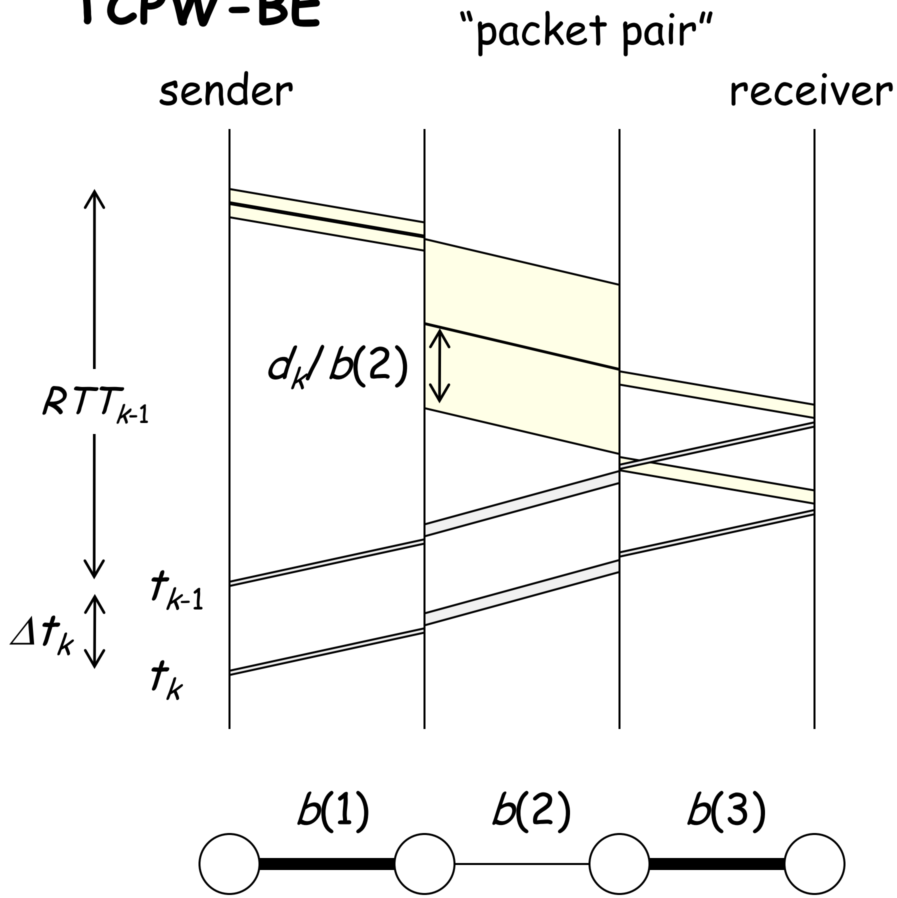
$$ssthresh = FSE * RTT_{\min}$$
$$cwnd = 1$$

- multiple versions according to FSE estimation methods

C.Casetti et al: "TCP Westwood: Bandwidth Estimation for Enhanced Transport over Wireless Links", ACM MOBICOM 2001.

Bandwidth Share Estimation

TCPW-BE



Bandwidth share: $b = \min_j (b(j))$

t_k: ack arrival time of the k-th packet

d_k: size of the k-th packet

$$\Delta t_k = t_k - t_{k-1} \approx \frac{d_k}{b}$$

$$b_k \approx \frac{d_k}{\Delta t_k}$$

moving average: $\hat{b}_k \rightarrow FSE$

C.Casetti et al: "TCP Westwood: Bandwidth Estimation for Enhanced Transport over Wireless Links", ACM MOBICOM 2001.

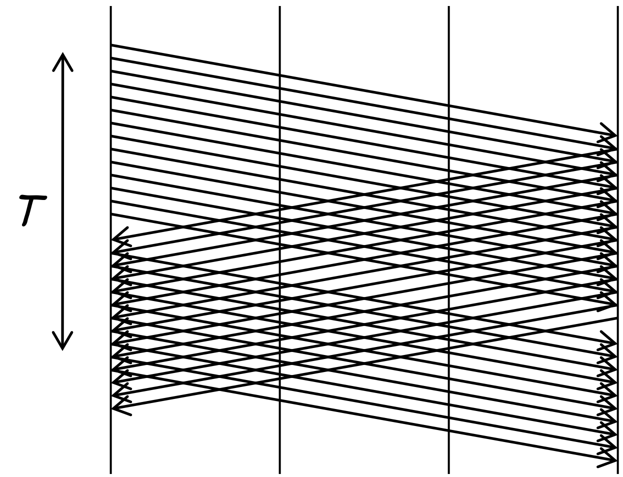
Rate Estimation

(reference) TCP-Vegas

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

expect rate

actual rate



TCPW-RE:

$$RE_k = \frac{\sum_{t_j > t_k - T} d_j}{T}$$

cwnd $\Rightarrow$ S=Σd_k

RTT$\Rightarrow$ T= $\Sigma \Delta t_k$

$$\hat{R}E_k \approx \frac{\sum d_k}{\sum \Delta t_k}$$

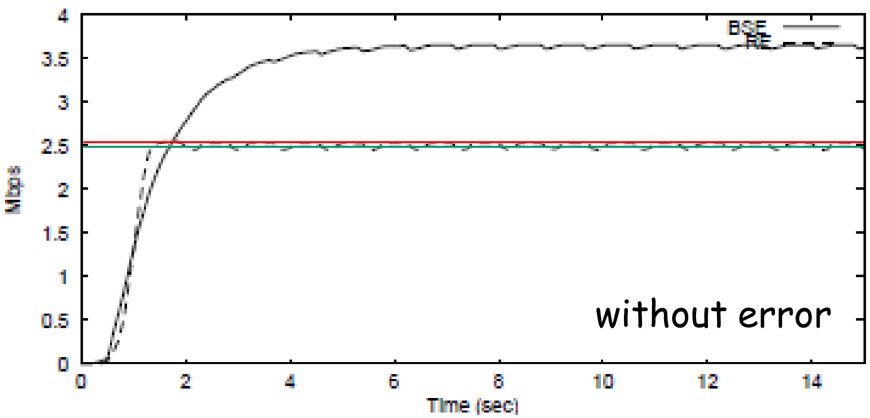
moving average: $\hat{R}E_k \rightarrow FSE$

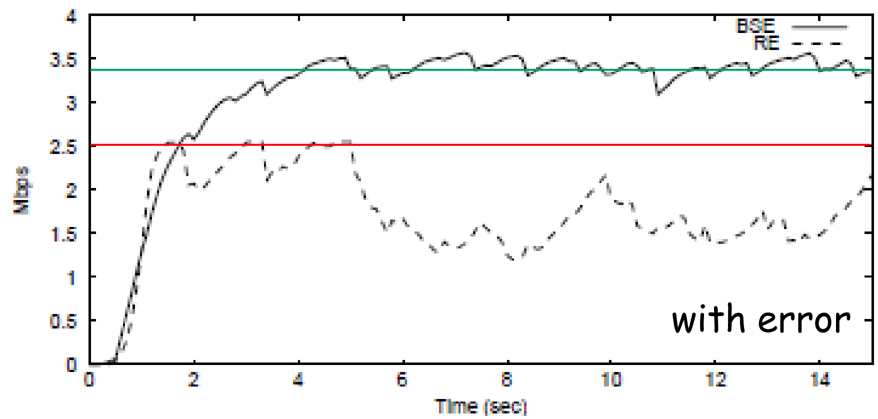
$T = n \cdot RTT$ (e.g. n=4)

M.Gerla et al: "TCP westwood with adaptive bandwidth estimation to improve ...", Comp. & Comm., 2004.

Comparison of BSE and RE

solid: BSE, dashed: RE, red: fair share, green: capacity





- BSE tends to overestimate (due to burstiness)
- RE tends to underestimate when losses occur

M.Gerla et al: "TCP westwood with adaptive bandwidth estimation to improve …", Comp. & Comm., 2004.

Adaptive Bandwidth Share Estimation

- BSE: overestimation, RE: underestimation
- difference lies in sampling period *T*

- large T when congested (BSE), small T when not congested (RE)

TCPW-ABSE:

$$T_k = \max\left(T_{\min},\ RTT \cdot \left(1 - \frac{\overbrace{\hat{Th}}^{\text{actual rate}} \cdot RTT_{\min}}{cwnd} \right) \right)$$

T_{min} : ACK arrival interval

$\hat{Th} \cdot RTT_{\min} < cwnd$ ⇨ congested ⇨ larger T_k

多数のパケットを送っても実レートが上がらない

M.Gerla et al: "TCP westwood with adaptive bandwidth estimation to improve …", Comp. & Comm., 2004.

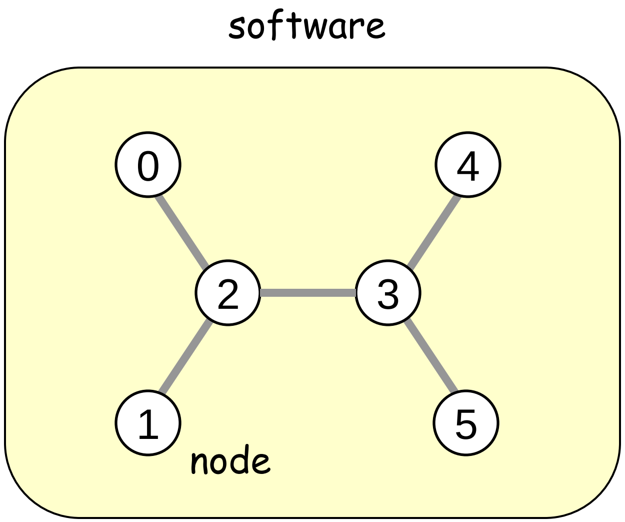
Network Simulation & Emulation

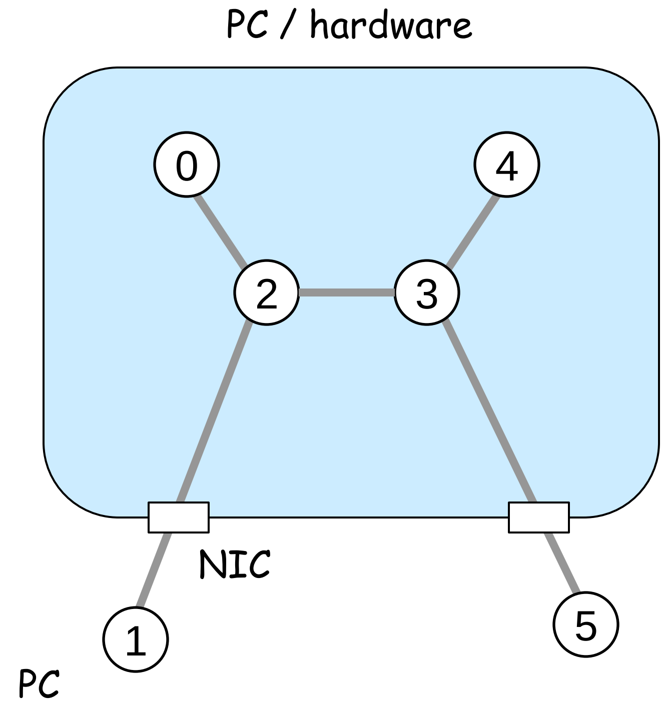
Networking Research

- Algorithm
- Theory (model)
- <u>Simulation</u>
- <u>Emulation</u>
- Implementation

Simulator & Emulator (1)

- simulation
- emulation





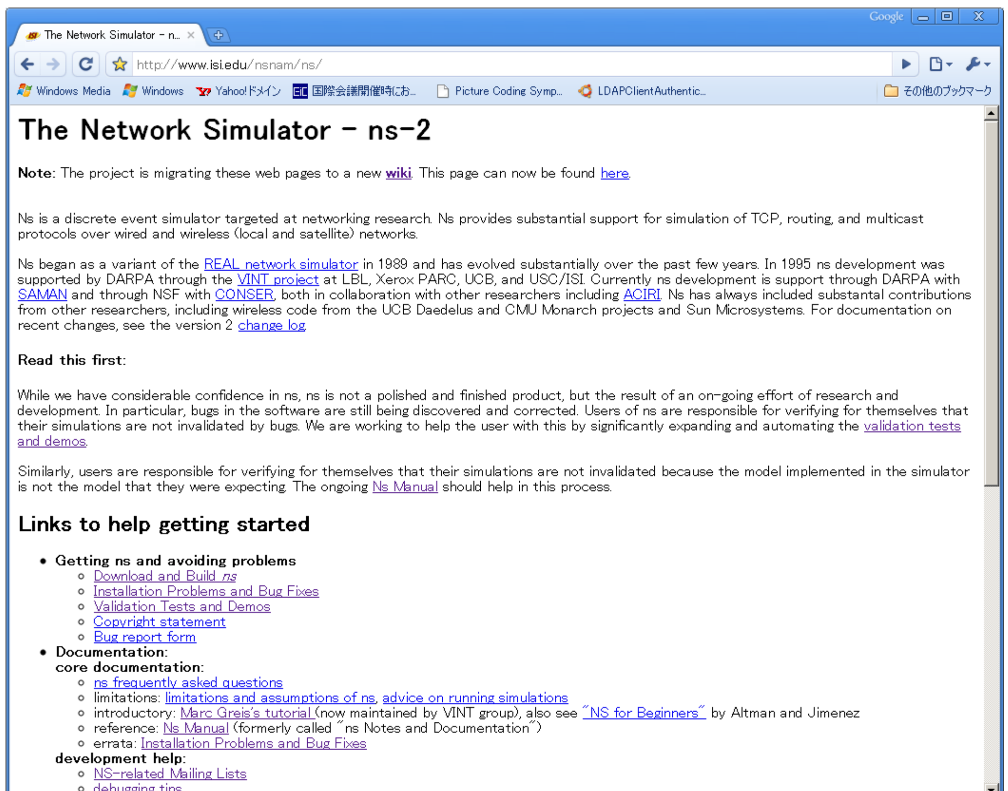
Simulator & Emulator (2)

simulator	emulator	URL
ns-2 (ns)	(nse)	http://www.isi.edu/nsnam/ns/
ns-3	nsc	http://www.nsnam.org/
OPNET		http://www.opnet.com/
Qualnet, GloMoSim	EXata	http://www.scalable-networks.com/
Scenargie		http://www.spacetime-eng.com/
	PacketStorm	http://www.packetstorm.com/
	Cloud	http://www.shunra.com/ve-cloud.php

ns-2

Ns-2 (1)

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



The Network Simulator - ns-2

Note: The project is migrating these web pages to a new **wiki**. This page can now be found here.

Ns is a discrete event simulator targeted at networking research. Ns provides substantial support for simulation of TCP, routing, and multicast protocols over wired and wireless (local and satellite) networks.

Ns began as a variant of the REAL network simulator in 1989 and has evolved substantially over the past few years. In 1995 ns development was supported by DARPA through the VINT project at LBL, Xerox PARC, UCB, and USC/ISI. Currently ns development is support through DARPA with SAMAN and through NSF with CONSER, both in collaboration with other researchers including ACIRI. Ns has always included substantal contributions from other researchers, including wireless code from the UCB Daedelus and CMU Monarch projects and Sun Microsystems. For documentation on recent changes, see the version 2 change log.

Read this first:

While we have considerable confidence in ns, ns is not a polished and finished product, but the result of an on-going effort of research and development. In particular, bugs in the software are still being discovered and corrected. Users of ns are responsible for verifying for themselves that their simulations are not invalidated by bugs. We are working to help the user with this by significantly expanding and automating the validation tests and demos.

Similarly, users are responsible for verifying for themselves that their simulations are not invalidated because the model implemented in the simulator is not the model that they were expecting. The ongoing Ns Manual should help in this process.

Links to help getting started

- **Getting ns and avoiding problems**
  - Download and Build *ns*
  - Installation Problems and Bug Fixes
  - Validation Tests and Demos
  - Copyright statement
  - Bug report form
- **Documentation:**
  core documentation:
  - ns frequently asked questions
  - limitations: limitations and assumptions of ns, advice on running simulations
  - introductory: Marc Greis's tutorial (now maintained by VINT group), also see "NS for Beginners" by Altman and Jimenez
  - reference: Ns Manual (formerly called "ns Notes and Documentation")
  - errata: Installation Problems and Bug Fixes

  development help:
  - NS-related Mailing Lists
  - debugging tips,

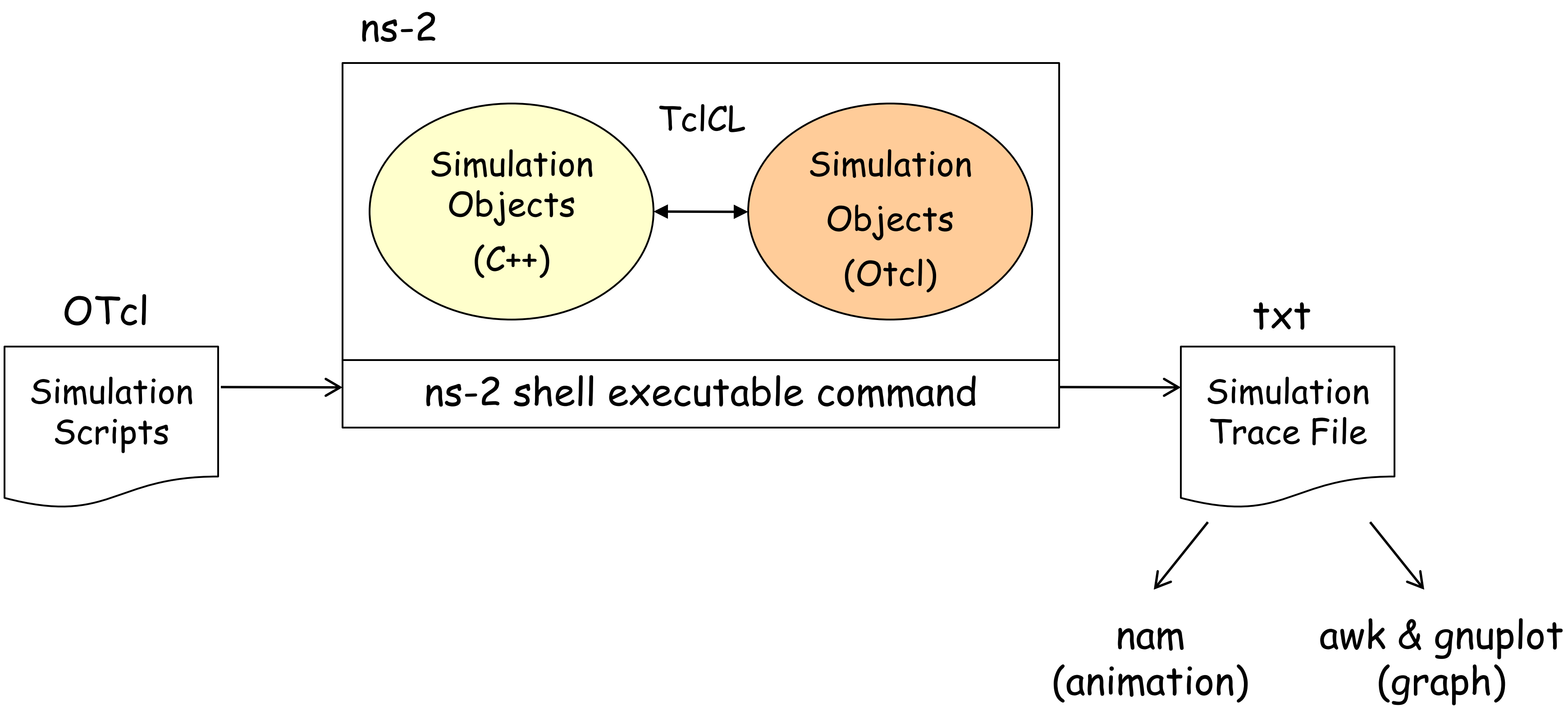
Ns-2 (2)

- download
  - 2.29 and later: http://sourceforge.net/projects/nsnam/
  - before 2.28: http://www.isi.edu/nsnam/dist/

Download “allinone”, expand、configure、and make
(Tcl/Tk, Otcl, TclCL, ns, nam)

Ns-2 (3)

- ns-2 Architecture



Ns-2 (4)

- Simulation scripts (*.tcl)

```
# initialization
# Simulator object
set ns [ new Simulator ]
# network topology
# definition of agents and apps
# procedure definition (e.g. finish)
proc finish () …
# event definition
$ns at 1.0 "$ftp start"
# simulation start
$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)

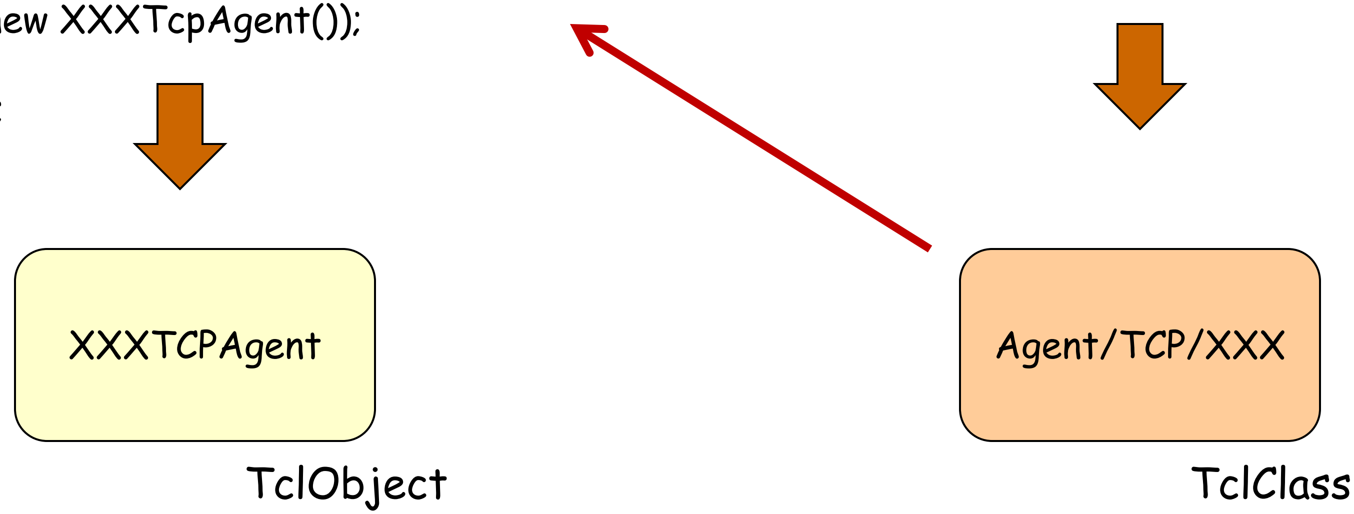
- Simulation Objects (C++/OTcl)

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]
```



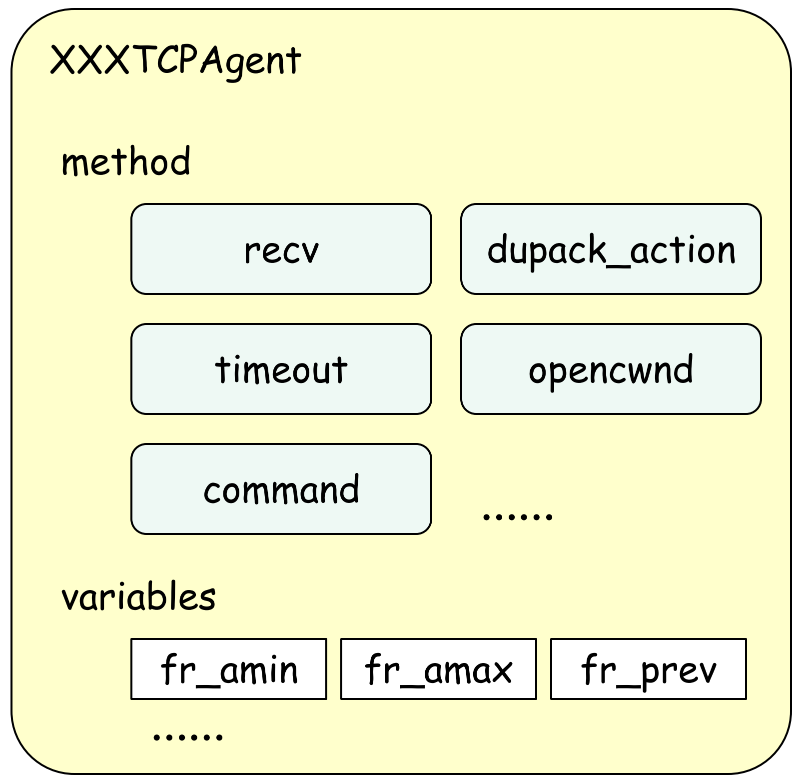
Ns-2 (6)

- 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)

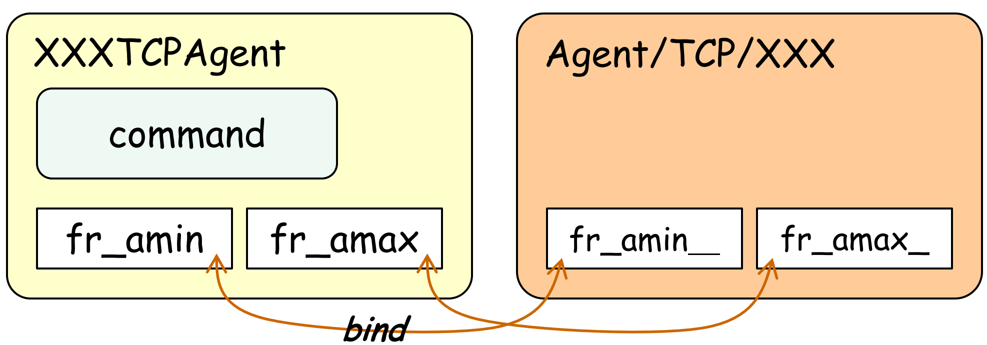
- Simulation Objects (C++/OTcl)

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)

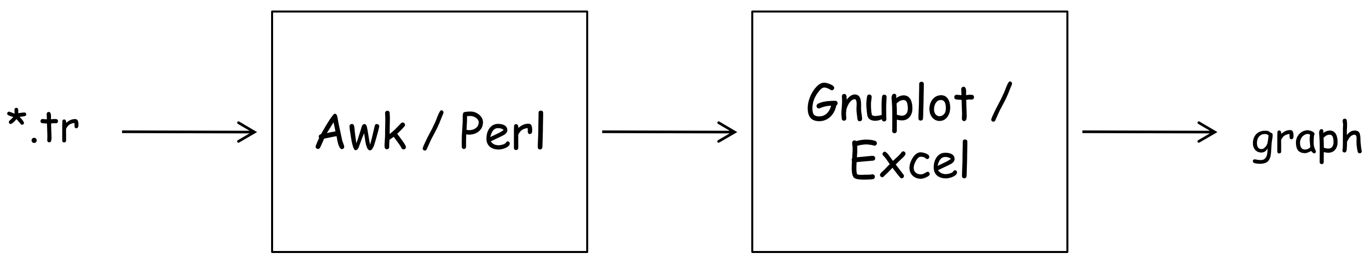
- Trace File (*.tr)

enqueue
dequeue
receive
drop

```
+ 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
```

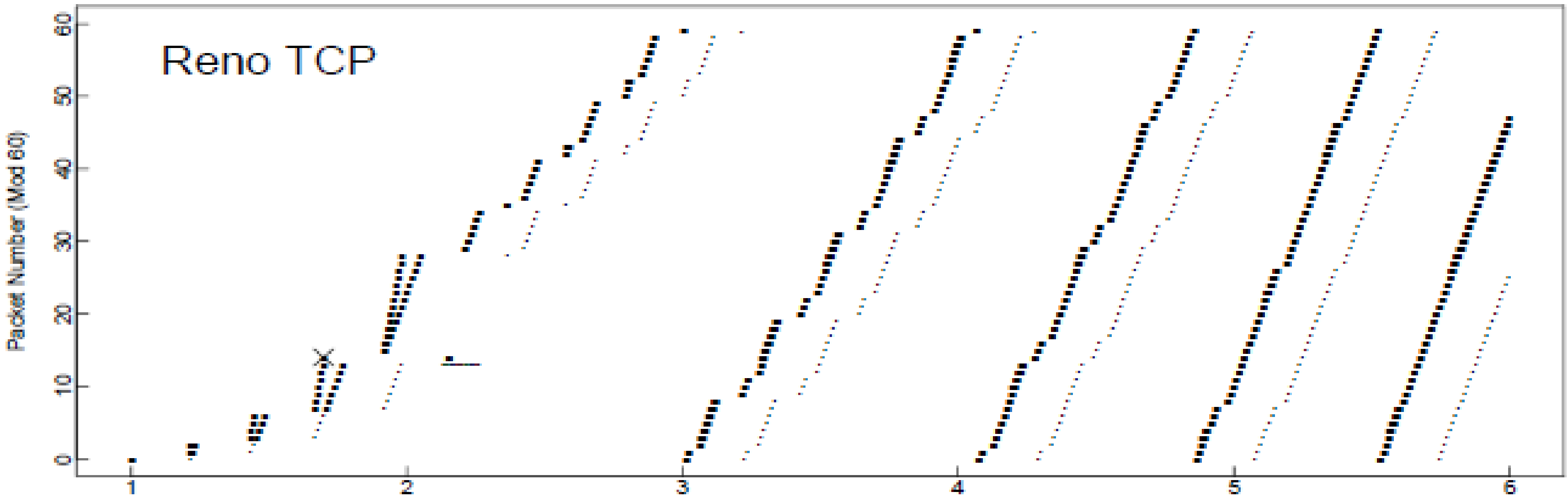
Ns-2 (9)

- Awk / Perl (script)
  - applied to trace files
  - generate graph files, e.g. for GnuPlot



Ns-2 (10)

- example



IEICE Network System Technical Group's Archive

- http://www.ieice.org/~ns/jpn/archives.html
  - 2009/8: ns-2 (summer school)
  - 2009/8: OPNET (summer school)
  - 2009/12: Qualnet (tutorial)

Googling "ns-2 tutorial" gives many sites

※ "ns-3" sites are increasing

ns-2 TCP-Linux

http://netlab.caltech.edu/projects/ns2tcplinux/

ns-2 TCP-Linux (1)

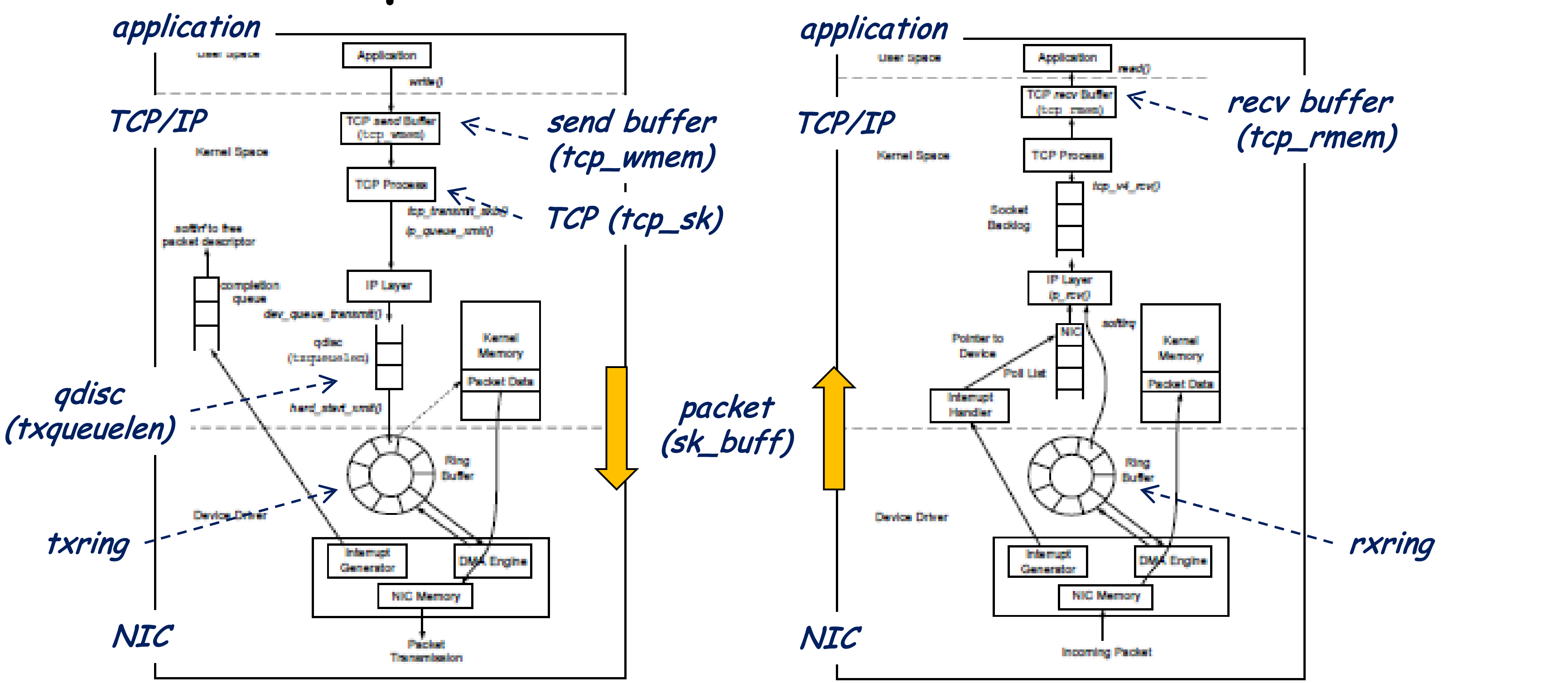
- ns-2 simulation using TCP implementation code in Linux kernel
  - bridge between implementations (Linux kernel) and simulations (ns-2)
    - fill a gap between implementation and simulation
  - verification of implementation codes

ns-2 TCP-Linux (2)

- TCPs implemented in Linux kernel (2.6.16-3)
  - TCP-Reno, TCP-Vegas, HighSpeed-TCP, Scalable-TCP, BIC-TCP, CUBIC-TCP (default), TCP-Westwood, H-TCP, TCP-Hybla
- TCPs to be implemented
  - TCP-Veno, TCP-LowPriority, Compound-TCP (Windows)

ns-2 TCP-Linux (3)

- TCP Implementation in Linux



Packet transmission

Packet reception

http://www.ece.virginia.edu/cheetah/documents/papers/TCPlinux.pdf

ns-2 TCP-Linux (4)

- Variables in tcp_sk

Name	Meaning	Equivalent in NS-2 *TCPAgent*
snd_ ssthresh	slow-start threshold	ssthresh_
snd_ cwnd	integer part of the congestion window	trunc(cwnd_)
snd_ cwnd_cnt	fraction of congestion window	trunc(cwnd_^2) %trunc(cwnd_)
icsk_ca_ priv	a 512-bit array to hold per-flow state for a congestion control algorithm	n/a
icsk_ca_ ops	a pointer to the congestion control algorithm interface	n/a

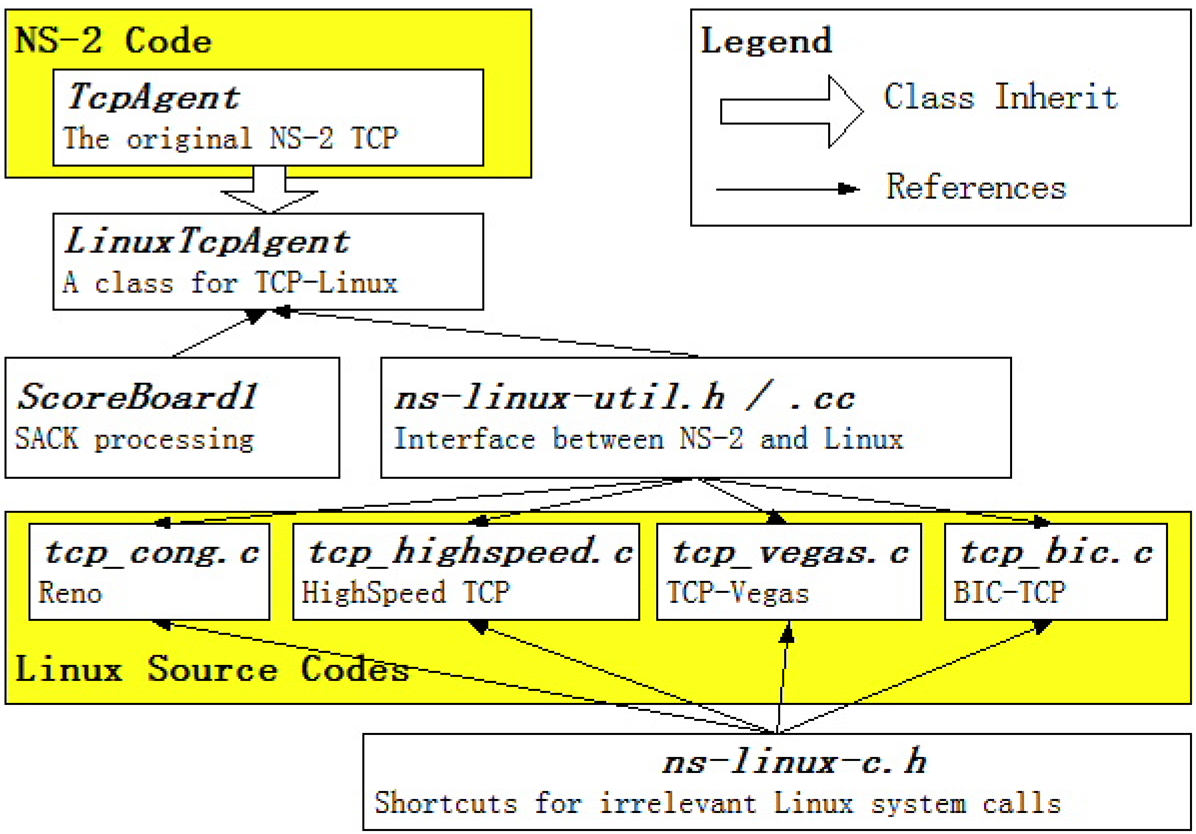
Congestion control modules:

cong_avoid: slow start & congestion avoidance
ssthresh: loss event handling
min_cwnd: fast retransmission

http://netlab.caltech.edu/projects/ns2tcplinux/

ns-2 TCP-Linux (5)

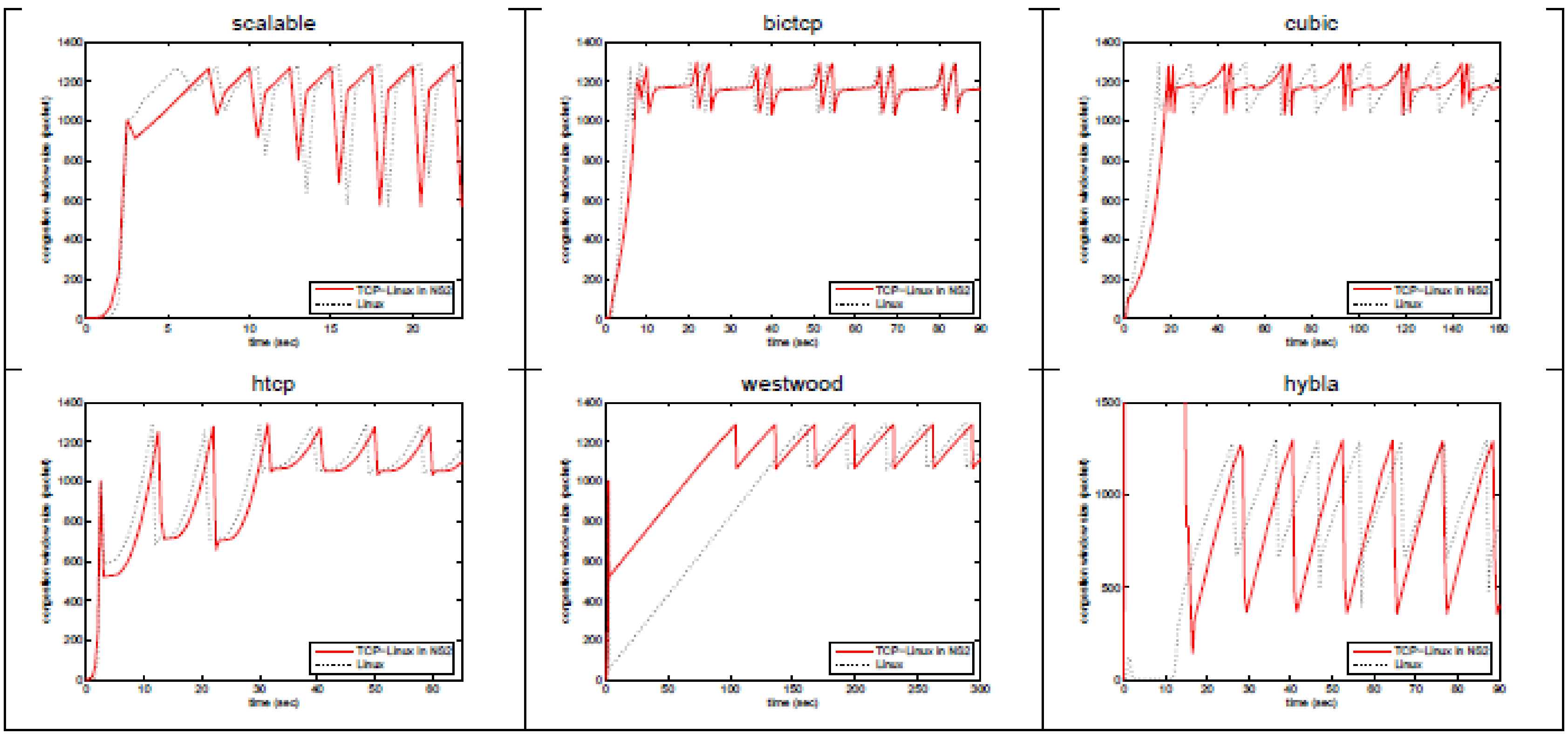
- Code structure



http://netlab.caltech.edu/projects/ns2tcplinux/

ns-2 TCP-Linux (6)

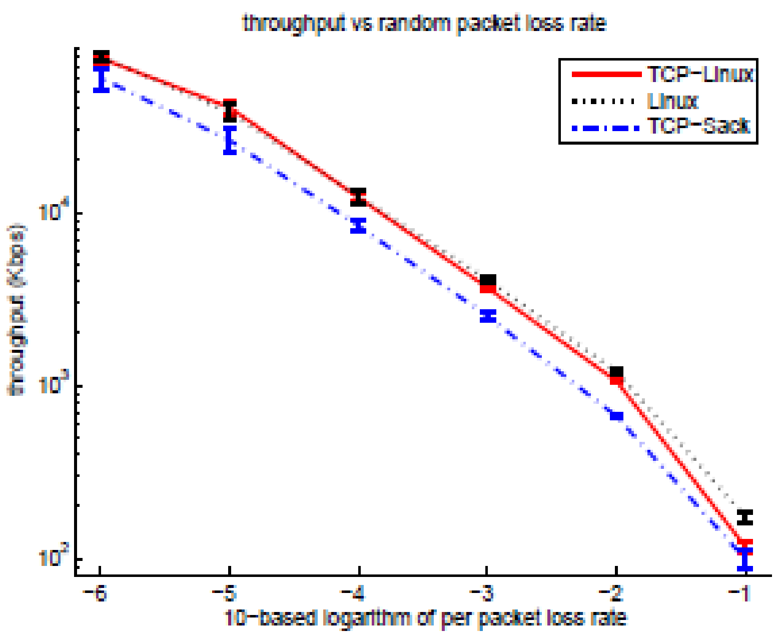
- Simulation (1) ns-2 & Linux



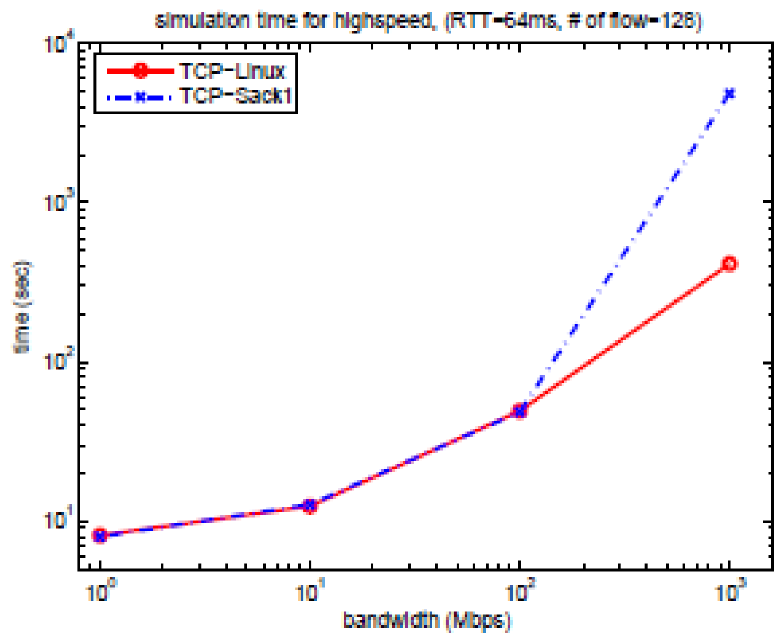
http://netlab.caltech.edu/projects/ns2tcplinux/

ns-2 TCP-Linux (7)

- Simulation (2) accuracy & speed



Accuracy



Speed

http://netlab.caltech.edu/projects/ns2tcplinux/

ns-3

ns-3 software overview

- ns-3 is written in C++, with bindings available for Python
  - simulation programs are C++ executables or Python programs
  - Python is often a glue language, in practice

- ns-3 is a GNU GPLv2-licensed project
- ns-3 lacks an integrated development / visualization environment (IDE)
- ns-3 is not backwards-compatible with ns-2

http://www.nsnam.org/docs/ns-3-overview.ppt

ns-3 development process

ns-3 is run as an open source project backed by research funding

- GPLv2 licensing
- open mailing lists
- uses standard tools (Mercurial, Bugzilla, Mediawiki, GNU/Linux development)
- ~20 maintainers worldwide

http://www.nsnam.org/docs/ns-3-overview.ppt

Available modules (ns-3.11 May 2011)

devices

bridge
spectrum
csma
tap-bridge
emu
uan
point-to-point
virtual-net-device
lte
wifi
mesh
wimax

applications
internet
network
core

energy
mpi
mobility
propagation

protocols

aodv
dsdv
olsr
click
nix-vector-routing

utilities

config-store
flow-monitor
netanim
stats
topology-read
visualizer

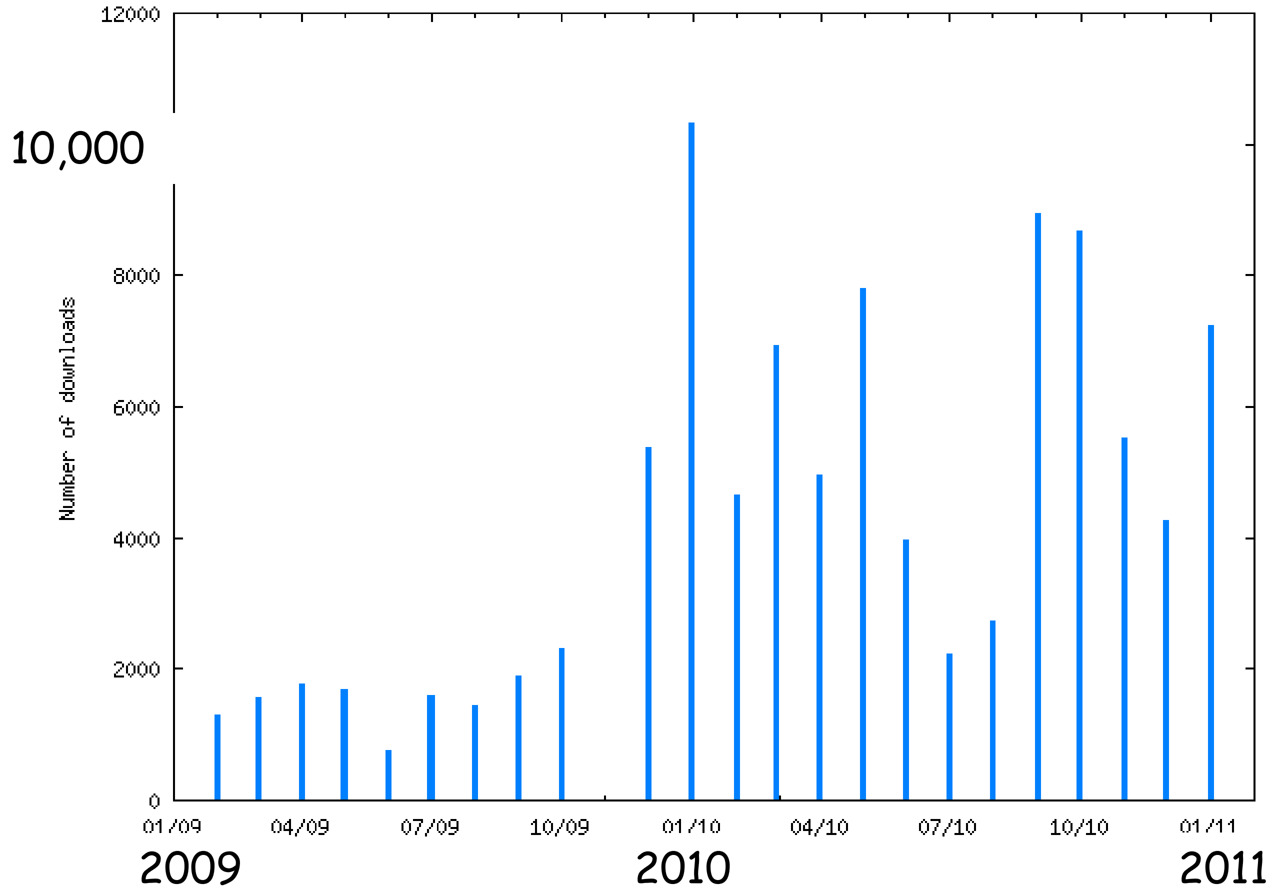
http://www.nsnam.org/docs/ns-3-overview.ppt

Analytics

Mailing list subscriptions:
• ns-3-users: 963
• ns-developers: 1176

Downloads:
• 6000/month in 2010

Number of ns-3 downloads/month



http://www.nsnam.org/docs/ns-3-overview.ppt

summary of ns-3 features

- modular, documented core
- C++ programs or (optionally) Python scripting
- alignment with real systems (sockets, device driver interfaces)

- emphasis on software integration
- virtualization and testbed integration are a priority (emulation modes)
- well-documented attribute system
- updated models

http://www.nsnam.org/docs/ns-3-overview.ppt

Resources

Web site:

http://www.nsnam.org

Mailing list:

http://mailman.isi.edu/mailman/listinfo/ns-developers

IRC: #ns-3 at freenode.net

Tutorial:

http://www.nsnam.org/docs/tutorial/tutorial.html

Code server:

http://code.nsnam.org

Wiki:

http://www.nsnam.org/wiki/index.php/Main_Page

"Architecture, design and source code comparison of ns-2 and ns-3 network simulators", 2010 Spring Simulation Multi-conference

http://portal.acm.org/citation.cfm?id=1878651