2.5 トランスポート層

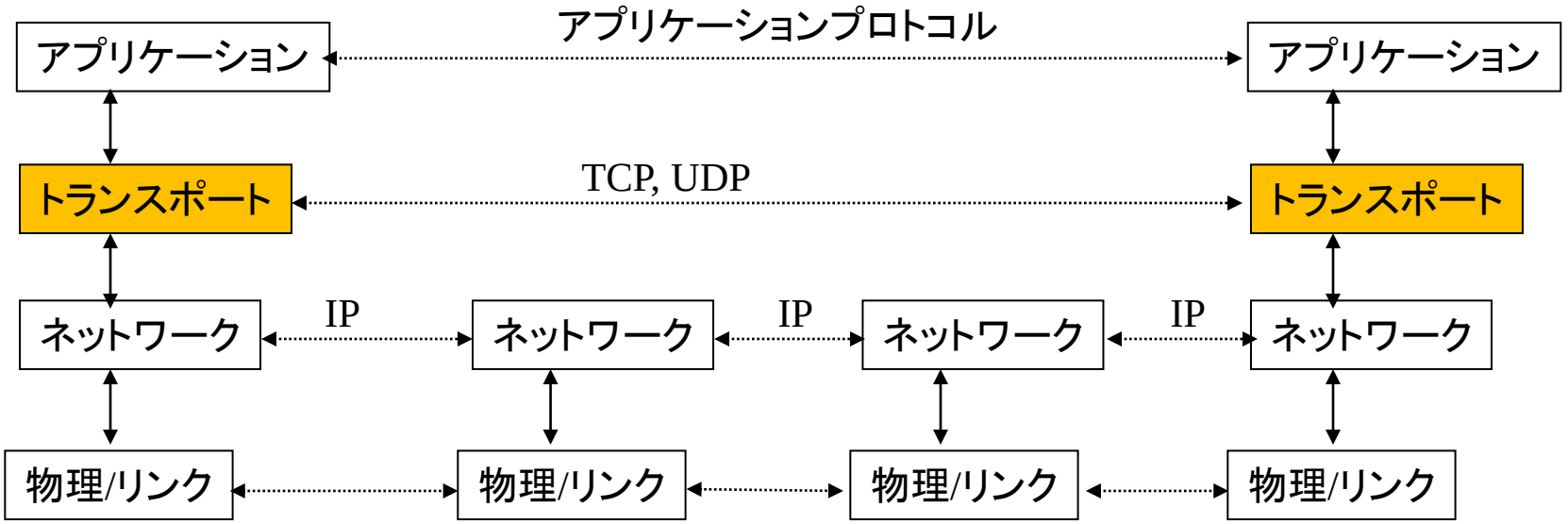


図1-12 TCP/IPの4レイヤモデル

トランスポートレイヤの仕事

- 計算機間での 良好な データのやり取りを実現する。
  - 誤りがないように
    - 再送
    - パリティー情報による自動再生(FEC; Forward Error Correction)
  - データを取りこぼさないように

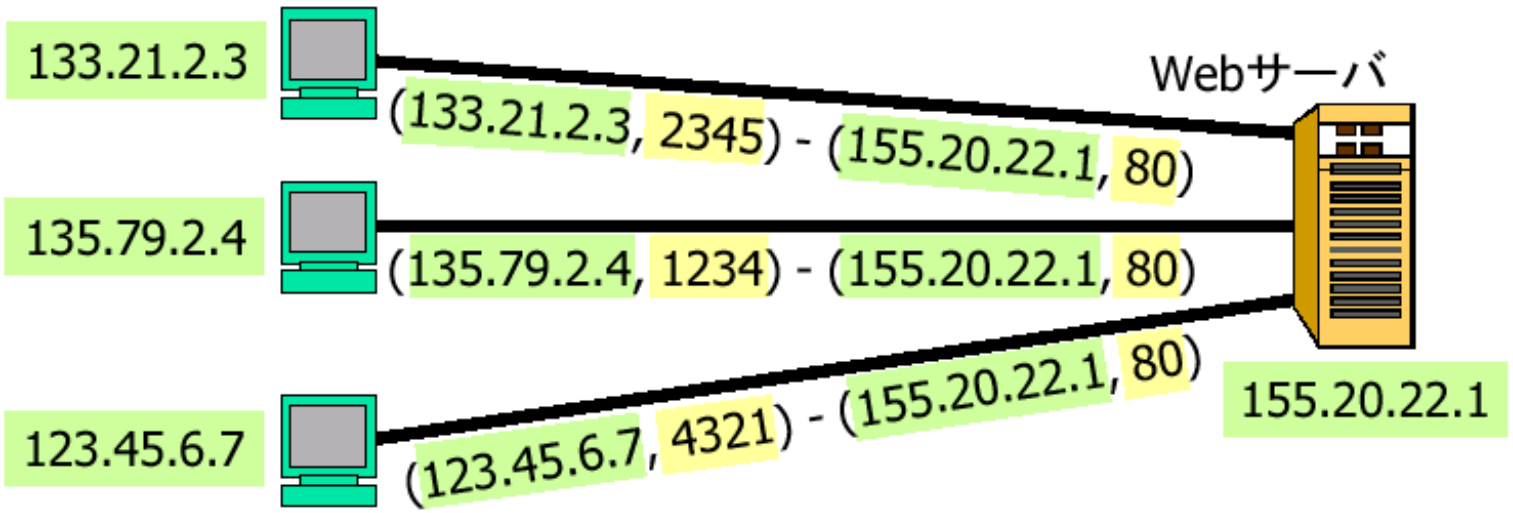
  (*) ファイルアクセスと同じ インターフェース を提供
- それ以外に欲しくなる機能
  - 並列データ転送
  - できるだけ高速に転送
  - ネットワークに “やさしく”
    - 道が混まないように
    - ネットワークは単純化、エンドホストが賢く

TCPとUDP

- TCP (Transmission Control Protocol)
  - コネクション型、ギャランティード
  - マルチキャスト・ブロードキャスト不可
- UDP (User Datagram Protocol)
  - コネクションレス、ベストエフォート
  - マルチキャスト・ブロードキャスト可

cf. IP (Internet Protocol)
  - コネクションレス、ベストエフォート

TCPコネクションの多重化と識別



Well-known Portの例

ポート 番号	用途(アプリケーション層プロトコル)
20, 21	FTP: ファイル転送プロトコル(20:データ，21:制御)
25	SMTP: 簡易メール配送プロトコル
53	Domain Name Service (DNS)
80	HTTP: ハイパーテキスト 転送プロトコル

TCP Port Allocation（RFC1700)

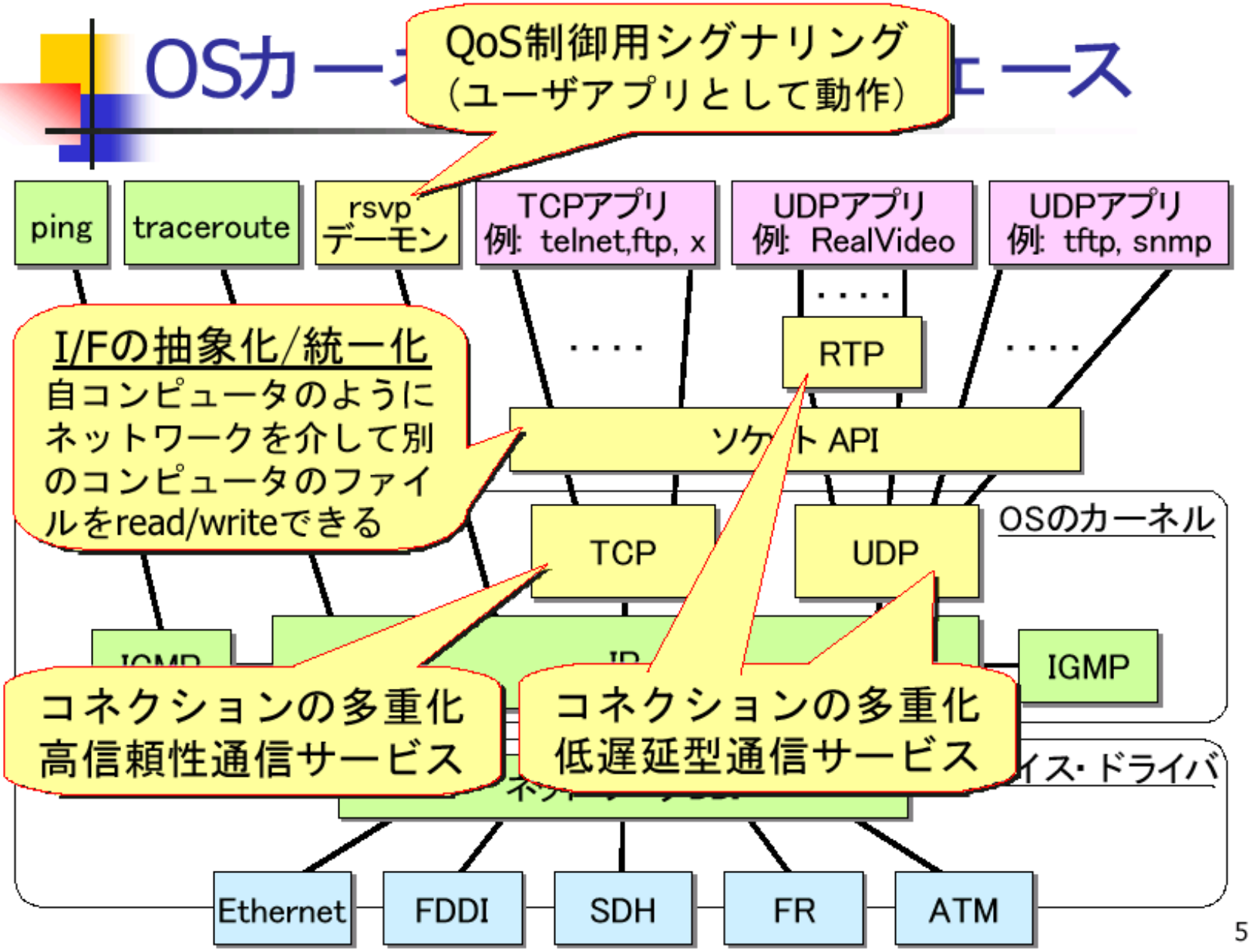
1. Well-Known Ports ; 0 - 1,023
2. Registered Ports ; 1,024 - 49,151
3. Dynamic and/or Private Ports ; 49,152 - 65,535

最新情報 :
ftp://ftp.isi.edu/in-notes/iana/assignments/port-numbers

TCP Features

1. **“*Stream*” Oriented** Data Transmission
   → Connection確立(*Three-way-handshake*)
2. Connection (“Stream”) Identifier ＝ ***“Socket”***
   {dst_IP_addr, dst_port, src_IP_addr, src_port}
3. “*Sequence Number*” ; 32 bits
   → バイト番号 : 0 ー (2^32-1)
   → 2^32 でSequence NumberがWrapされる
4. ***“Full-Duplex”***での通信
5. **Acknowledgement (ACK) ;**
   → 次に受信すべきバイト番号(SN)の通知
6. エラー回復: セグメント再送(Segment retransmission)
   by *Time-out, Dupilicated-ACK*
7. ***“Sliding Window Control”*** を用いたデータ転送制御
   (*) Window_size ≦ 65,535 Bytes

OSカーネルインタフェース



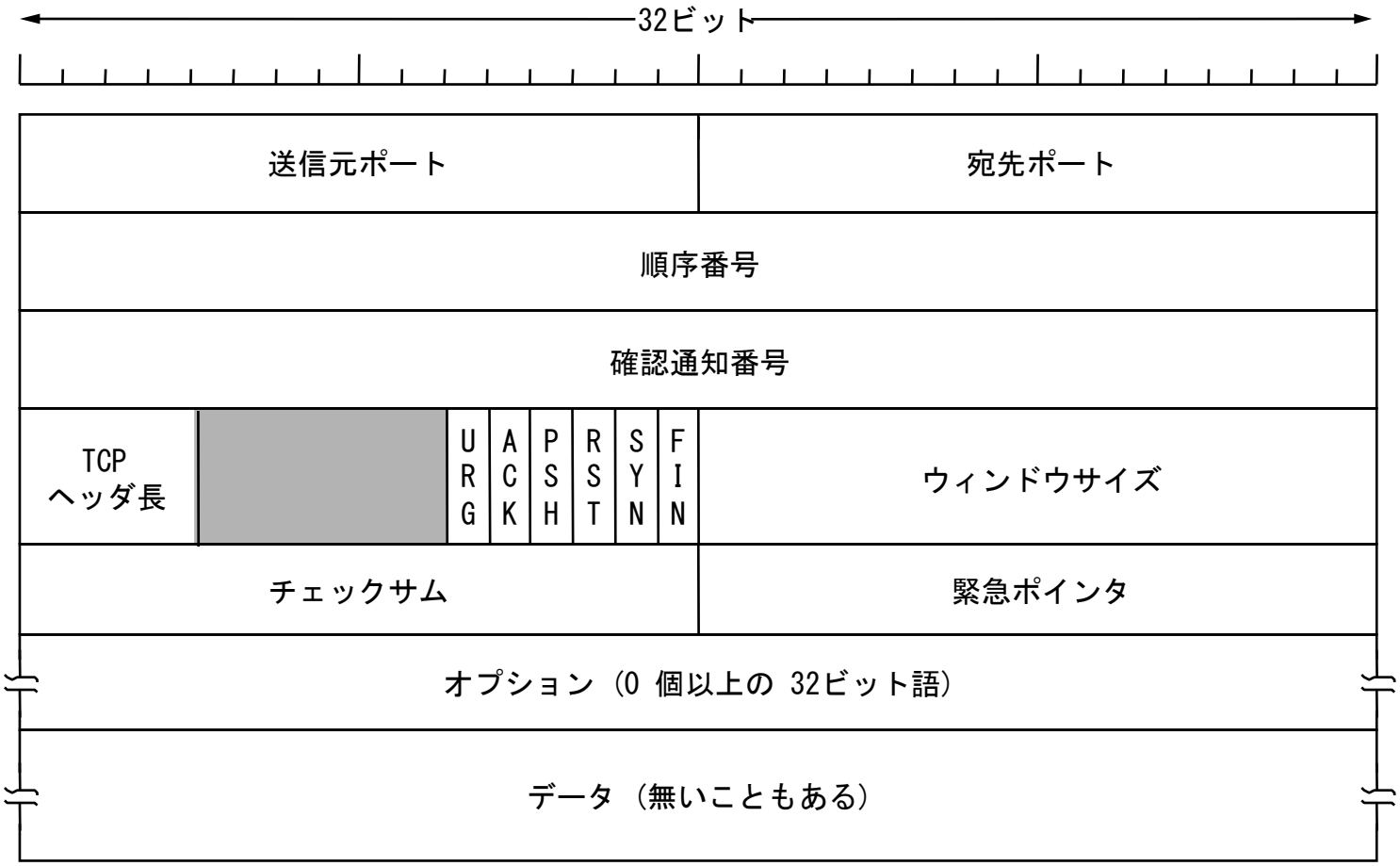
TCP Well-Known Ports

Port Number	Keyword	Application
5	rje	Remote Job Entry
20	ftp-data	File Transfer [Default data]
21	ftp	File Transfer [Control]
23	telnet	Telnet
25	smtp	Simple Management Protocol
39	rlp	Resource Location Protocol
53	domain	Domain Name Server
63	whois++	Whois++
67	bootp	Bootstrap Protocol Server
69	tftp	Trivial File Transfer
70	gopher	Gopher
79	finger	Finger
80	http	World Wide Web HTTP
110	pop3	Post Office Protocol - Version 3
111	sunrpc	SUN Remote Procedure Call
119	nntp	Network News Transfer Protocol

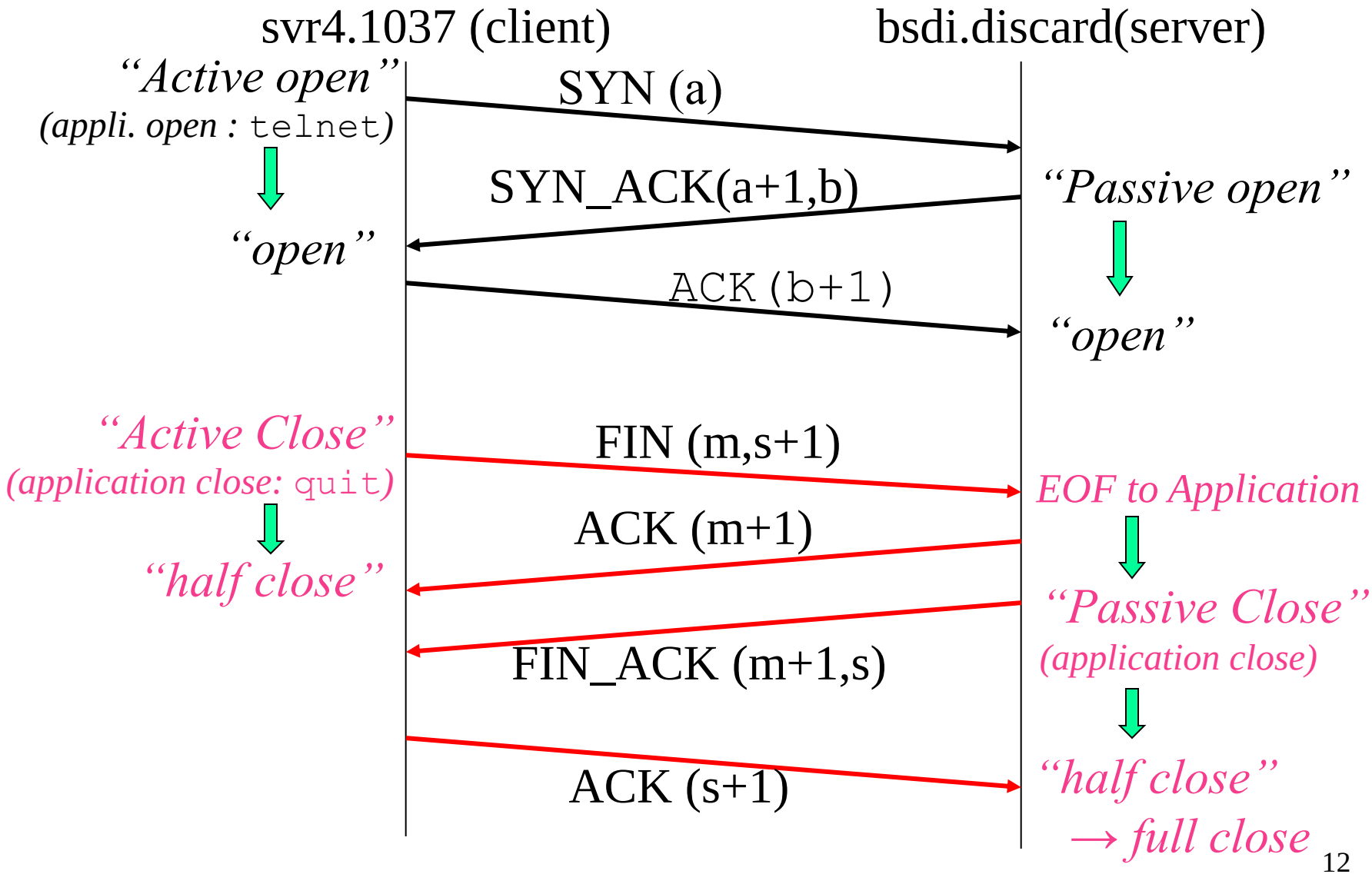
ポート番号の例

ポート番号	プロトコル名	アプリケーション
21	FTP	ファイル転送
23	telnet	遠隔ログイン
25	SMTP	電子メール
53	DNS	ドメイン名
80	HTTP	WWW
110	POP-3	電子メール読み出し
143	IMAP	電子メール読み出し
443	HTTPS	セキュアWWW

TCPセグメント

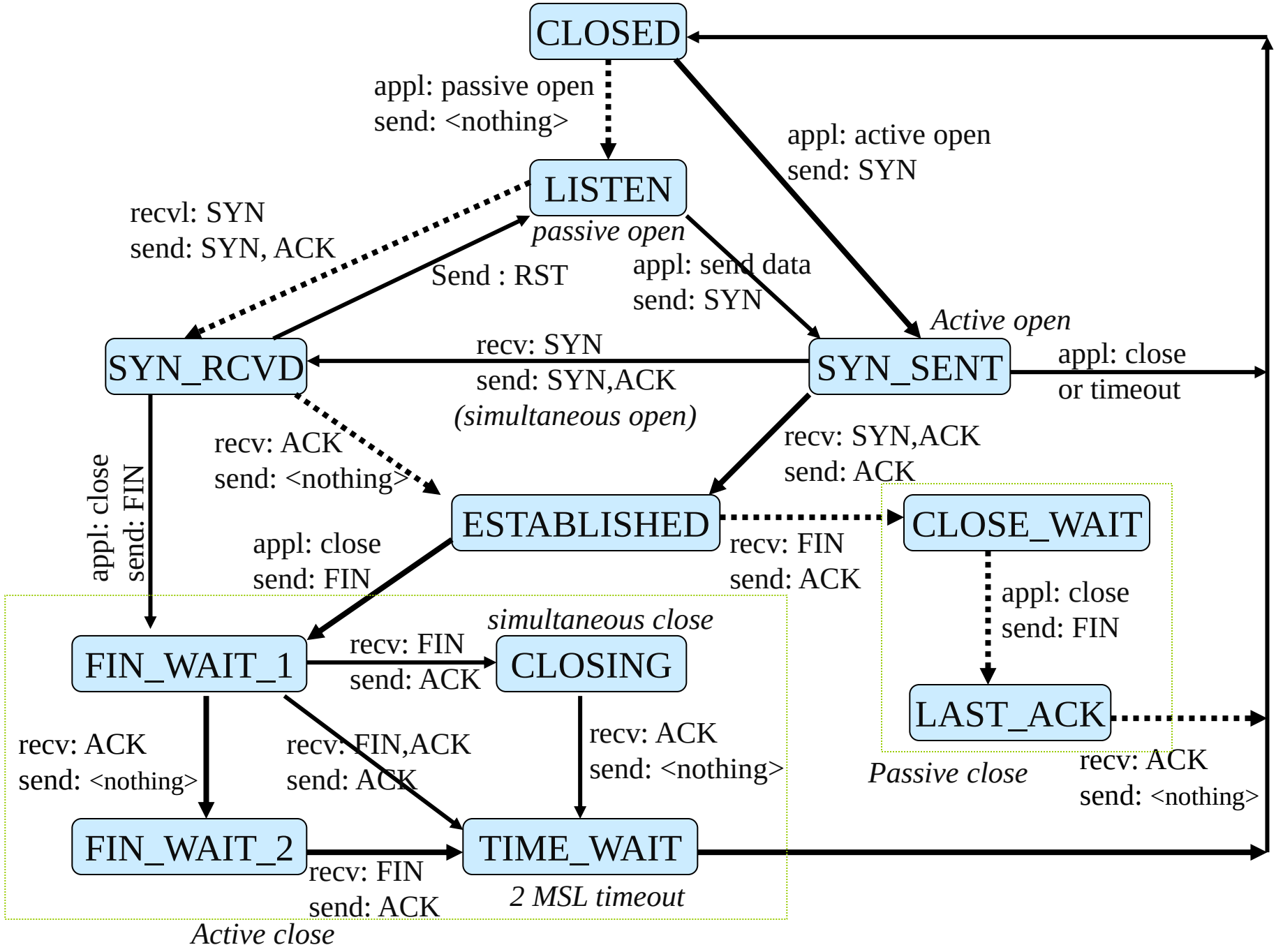

32ビット
送信元ポート
宛先ポート
順序番号
確認通知番号
TCP
ヘッダ長
URG
ACK
PSH
RST
SYN
FIN
ウィンドウサイズ
チェックサム
緊急ポインタ
オプション（0 個以上の 32ビット語）
データ（無いこともある）

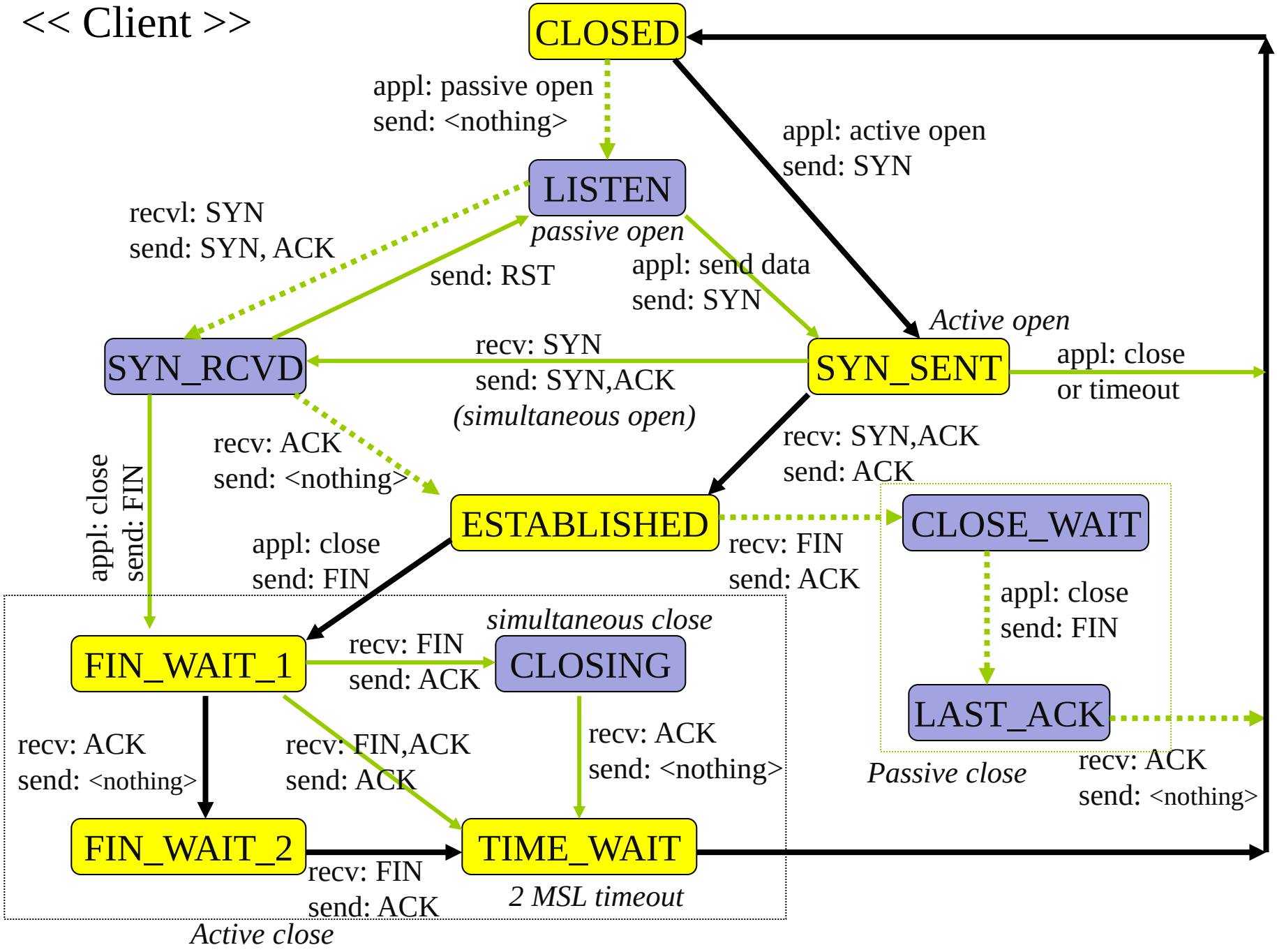
TCP Connection確立/開放

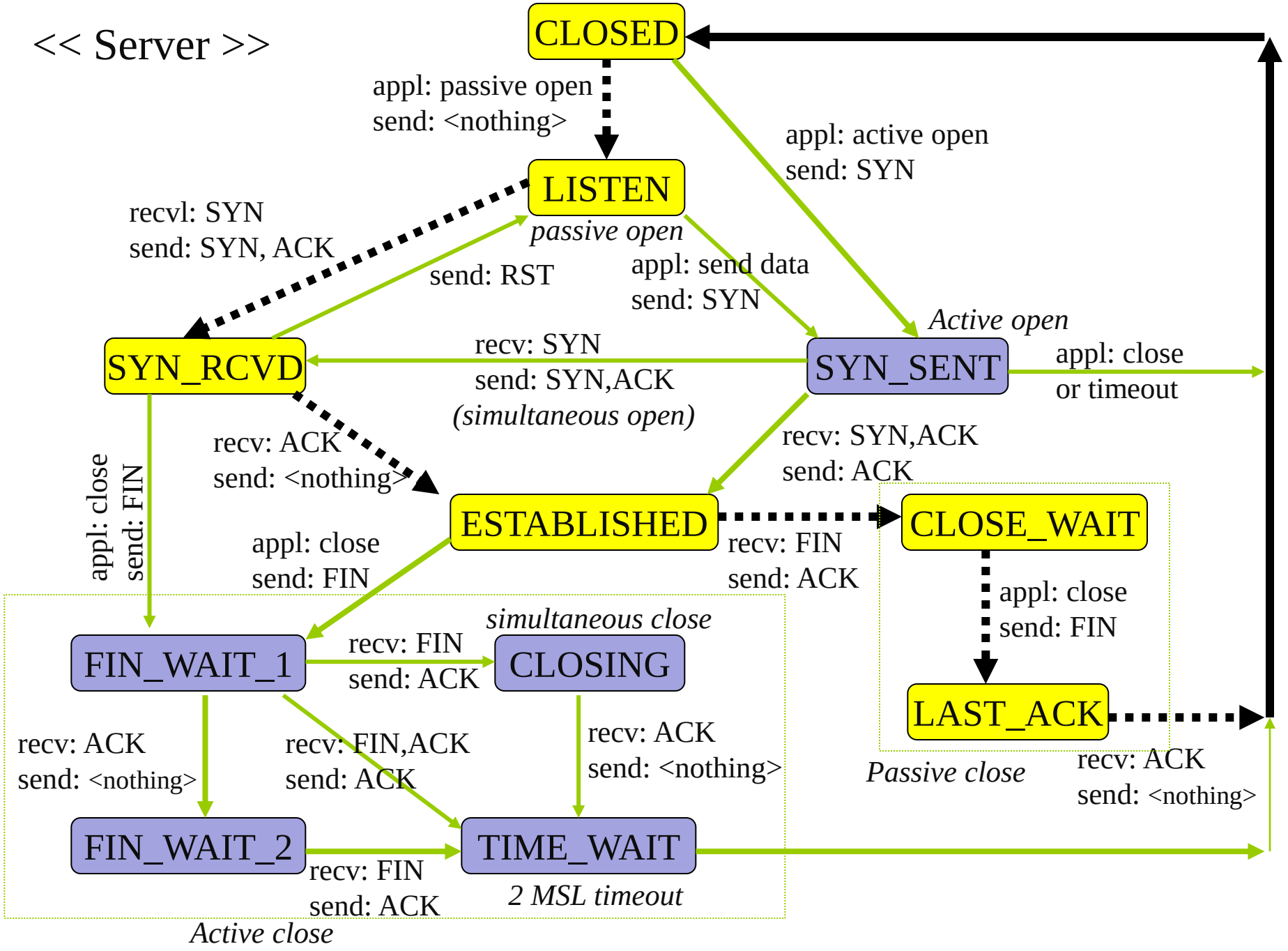


Three Way Handshalking

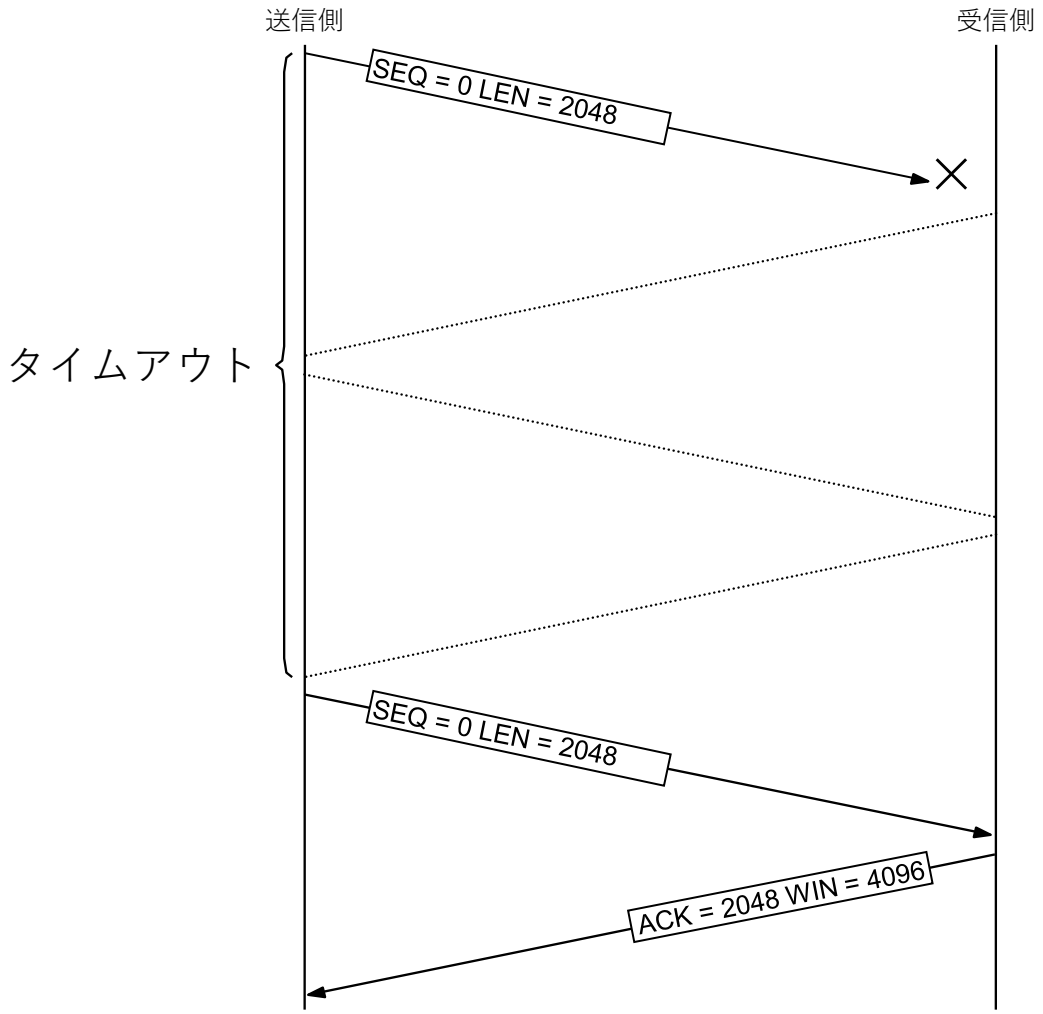
- エラー が発生する可能性が存在する。
- 2つの ノードの間での それぞれのノードが持つ 情報の「同期」を確立させたい。
- 3つのパケット(＝メッセージ)の 送受信で、情報の同期が確保・保証される(必要十分条件)。


CLOSED
appl: passive open
send: <nothing>
appl: active open
send: SYN
LISTEN
passive open
recvl: SYN
send: SYN, ACK
Send : RST
appl: send data
send: SYN
Active open
SYN_RCVD
recv: SYN
send: SYN,ACK
(simultaneous open)
SYN_SENT
appl: close
or timeout
recv: ACK
send: <nothing>
recv: SYN,ACK
send: ACK
appl: close
send: FIN
ESTABLISHED
CLOSE_WAIT
appl: close
send: FIN
recv: FIN
send: ACK
simultaneous close
FIN_WAIT_1
recv: FIN
send: ACK
CLOSING
LAST_ACK
recv: ACK
send: <nothing>
recv: FIN,ACK
send: ACK
recv: ACK
send: <nothing>
recv: ACK
send: <nothing>
Passive close
FIN_WAIT_2
TIME_WAIT
recv: FIN
send: ACK
2 MSL timeout
Active close

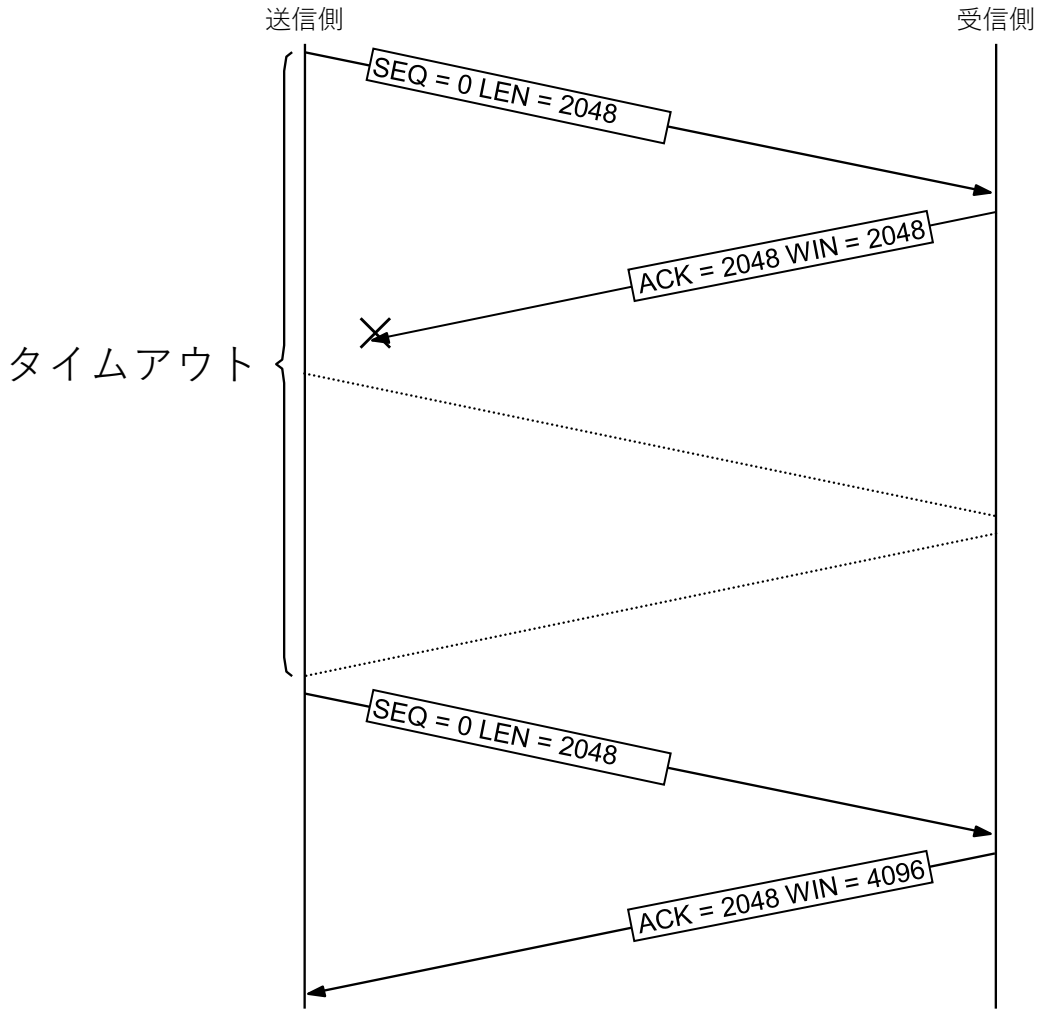

<< Client >>
CLOSED
appl: passive open
send: <nothing>
appl: active open
send: SYN
LISTEN
passive open
recvl: SYN
send: SYN, ACK
send: RST
appl: send data
send: SYN
Active open
SYN_RCVD
recv: SYN
send: SYN,ACK
(simultaneous open)
SYN_SENT
appl: close
or timeout
recv: ACK
send: <nothing>
recv: SYN,ACK
send: ACK
appl: close
send: FIN
ESTABLISHED
CLOSE_WAIT
appl: close
send: FIN
recv: FIN
send: ACK
simultaneous close
FIN_WAIT_1
recv: FIN
send: ACK
CLOSING
LAST_ACK
recv: ACK
send: <nothing>
recv: FIN,ACK
send: ACK
recv: ACK
send: <nothing>
recv: ACK
send: <nothing>
Passive close
FIN_WAIT_2
TIME_WAIT
recv: FIN
send: ACK
2 MSL timeout
Active close


<< Server >>
CLOSED
appl: passive open
send: <nothing>
appl: active open
send: SYN
LISTEN
passive open
recvl: SYN
send: SYN, ACK
send: RST
appl: send data
send: SYN
Active open
SYN_RCVD
recv: SYN
send: SYN,ACK
(simultaneous open)
SYN_SENT
appl: close
or timeout
recv: ACK
send: <nothing>
recv: SYN,ACK
send: ACK
appl: close
send: FIN
ESTABLISHED
CLOSE_WAIT
appl: close
send: FIN
recv: FIN
send: ACK
simultaneous close
FIN_WAIT_1
recv: FIN
send: ACK
CLOSING
LAST_ACK
recv: ACK
send: <nothing>
recv: FIN,ACK
send: ACK
recv: ACK
send: <nothing>
recv: ACK
send: <nothing>
Passive close
FIN_WAIT_2
TIME_WAIT
recv: FIN
send: ACK
2 MSL timeout
Active close

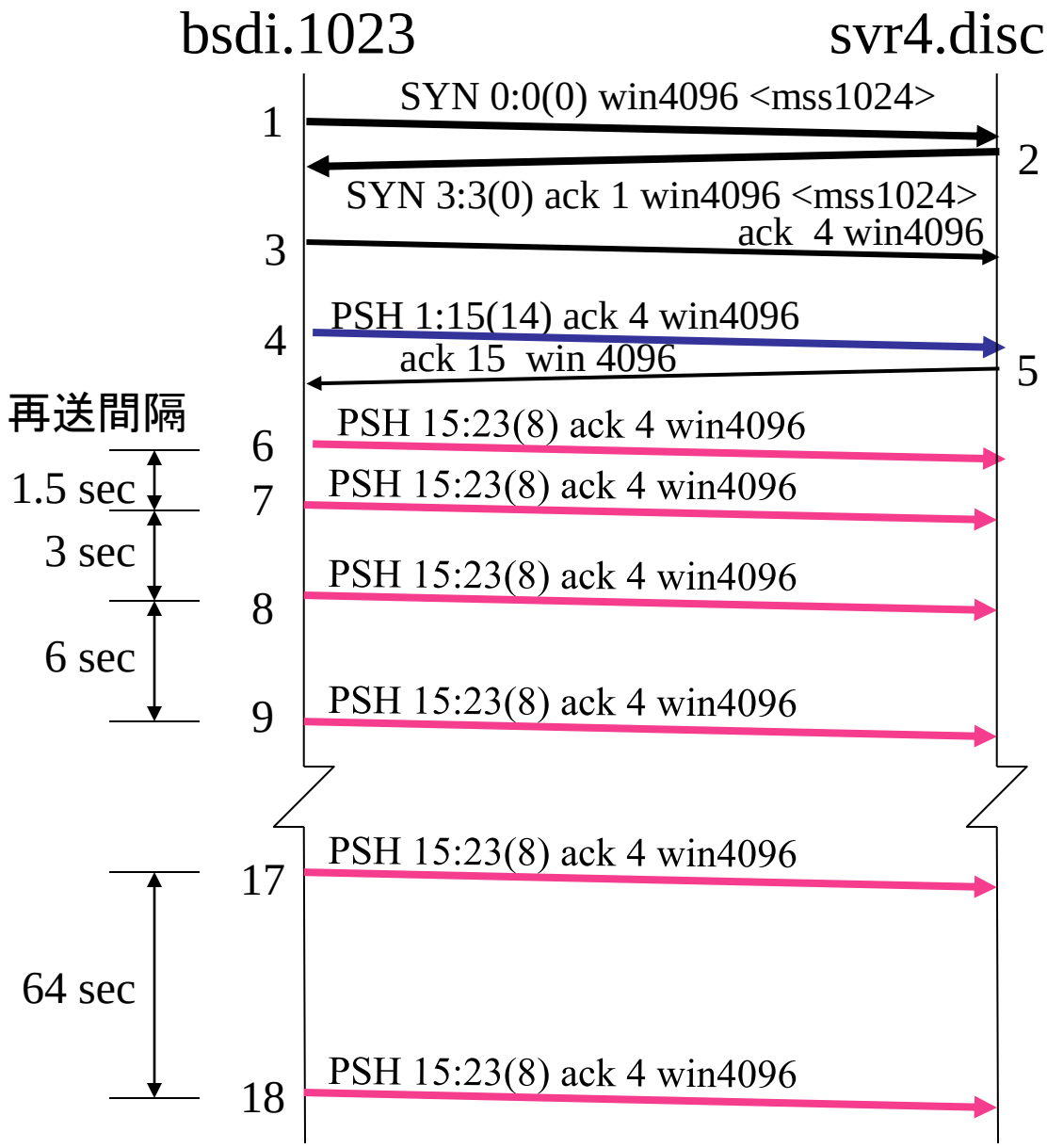
TCPの誤り制御



ACKが失われた場合


送信側
受信側
SEQ = 0 LEN = 2048
ACK = 2048 WIN = 2048
タイムアウト
SEQ = 0 LEN = 2048
ACK = 2048 WIN = 4096

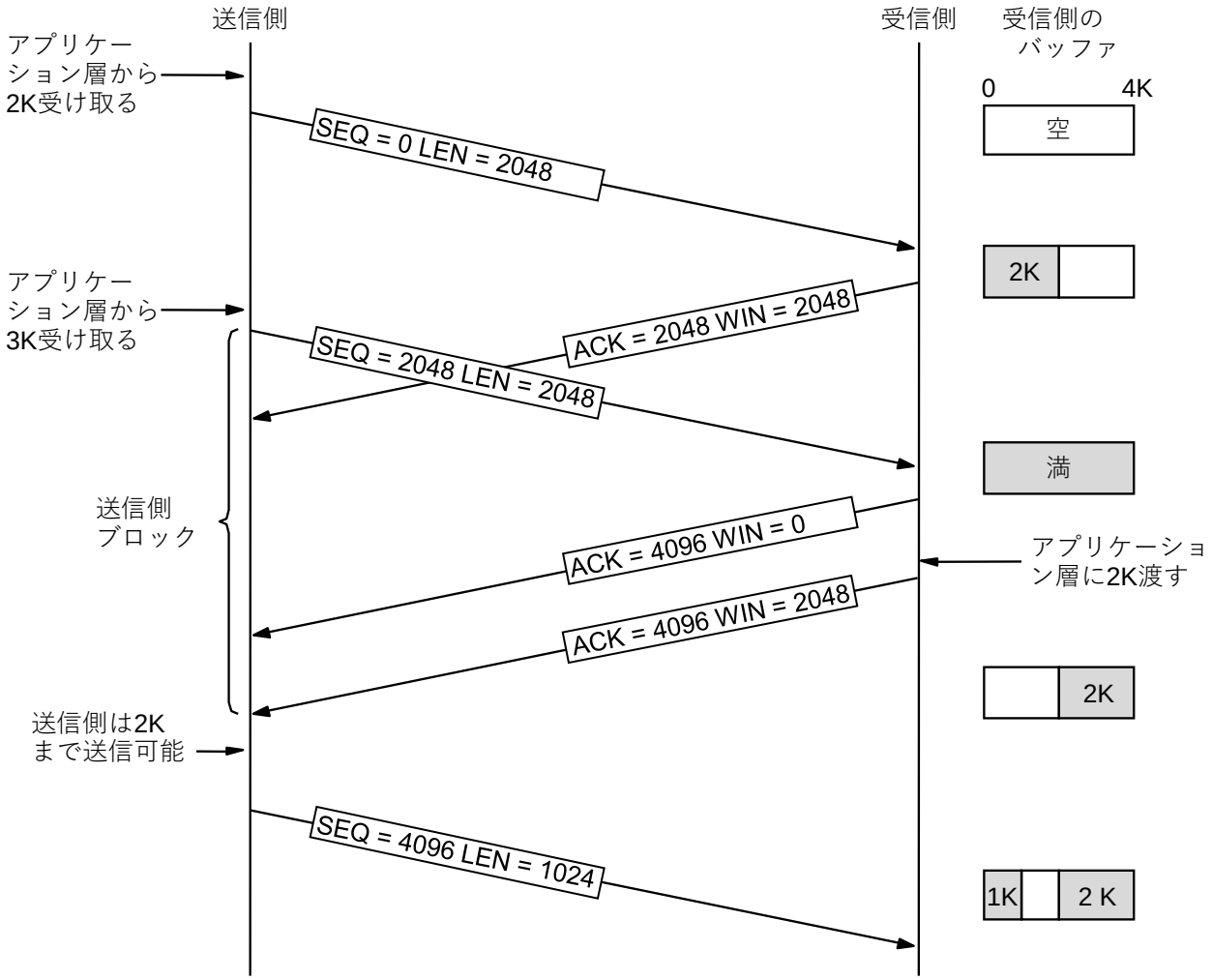
RTO Expired Retransmission



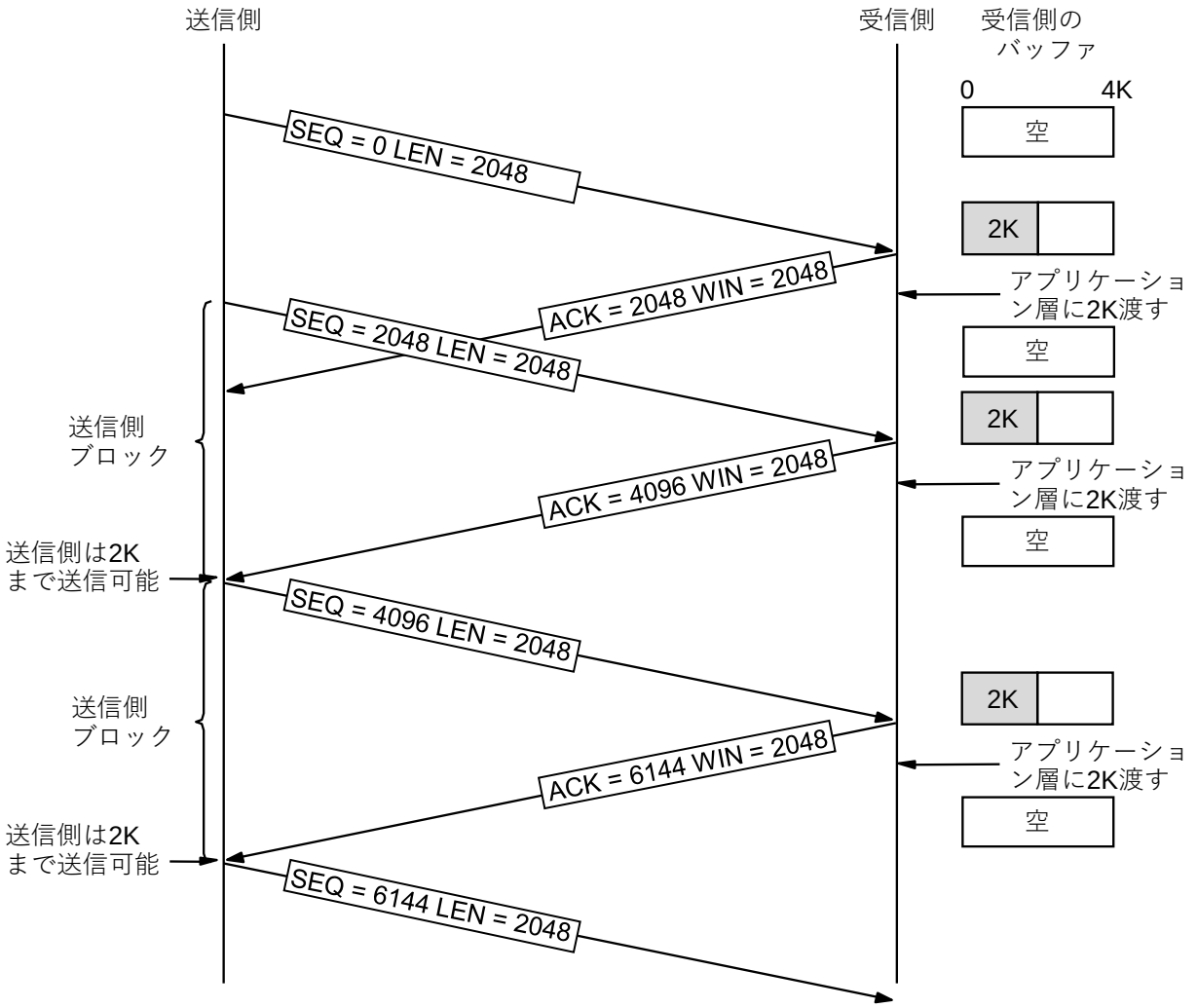
再送トライ (RTO; 再送タイマ)

```
RTO = 1.5 sec /* 変更可能*/
for ( 9 minutes)
 {
 if ( RTO expired)
  {
   retransmission;
   RTO=RTO x 2;
   RTO=min{64sec, RTO};
  }
 }
end /* 諦める */
```

TCPのフロー制御

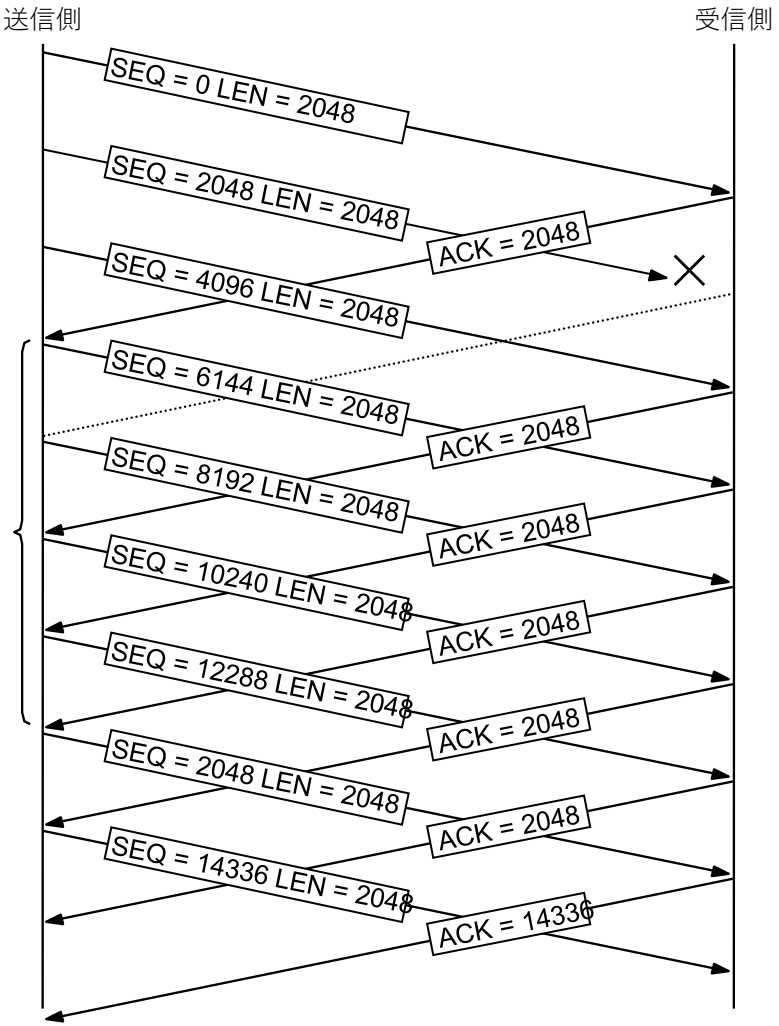

送信側
受信側
受信側の
バッファ
アプリケー
ション層から
2K受け取る
0
4K
空
SEQ = 0 LEN = 2048
2K
アプリケー
ション層から
3K受け取る
ACK = 2048 WIN = 2048
SEQ = 2048 LEN = 2048
満
送信側
ブロック
ACK = 4096 WIN = 0
アプリケーショ
ン層に2K渡す
ACK = 4096 WIN = 2048
2K
送信側は2K
まで送信可能
SEQ = 4096 LEN = 1024
1K
2 K

アプリケーション層が高速な場合



Fast Retransmit

重複するACKが
3つ続けて届いた
らタイムアウト
を待たずに再送



トランスポートレイヤの仕事

- 計算機間での 良好な データのやり取りを実現する。
  - 誤りがないように
    - 再送
    - パリティー情報による自動再生(FEC; Forward Error Correction)
  - データを取りこぼさないように

  (*) ファイルアクセスと同じ インターフェース を提供
- それ以外に欲しくなる機能
  - 並列データ転送
  - できるだけ 高速に転送
  - ネットワークに “やさしく”
    - 道が混まないように
    - ネットワークは単純化、エンドホストが賢く

連続送信可能ウィンドウサイズ

- ACKが戻ってくるよりウィンドウサイズが大きければ連続的に送信可能

  ➔ パイプライン状の投機的転送

例：100Mbps, 片道20km≒0.1ms

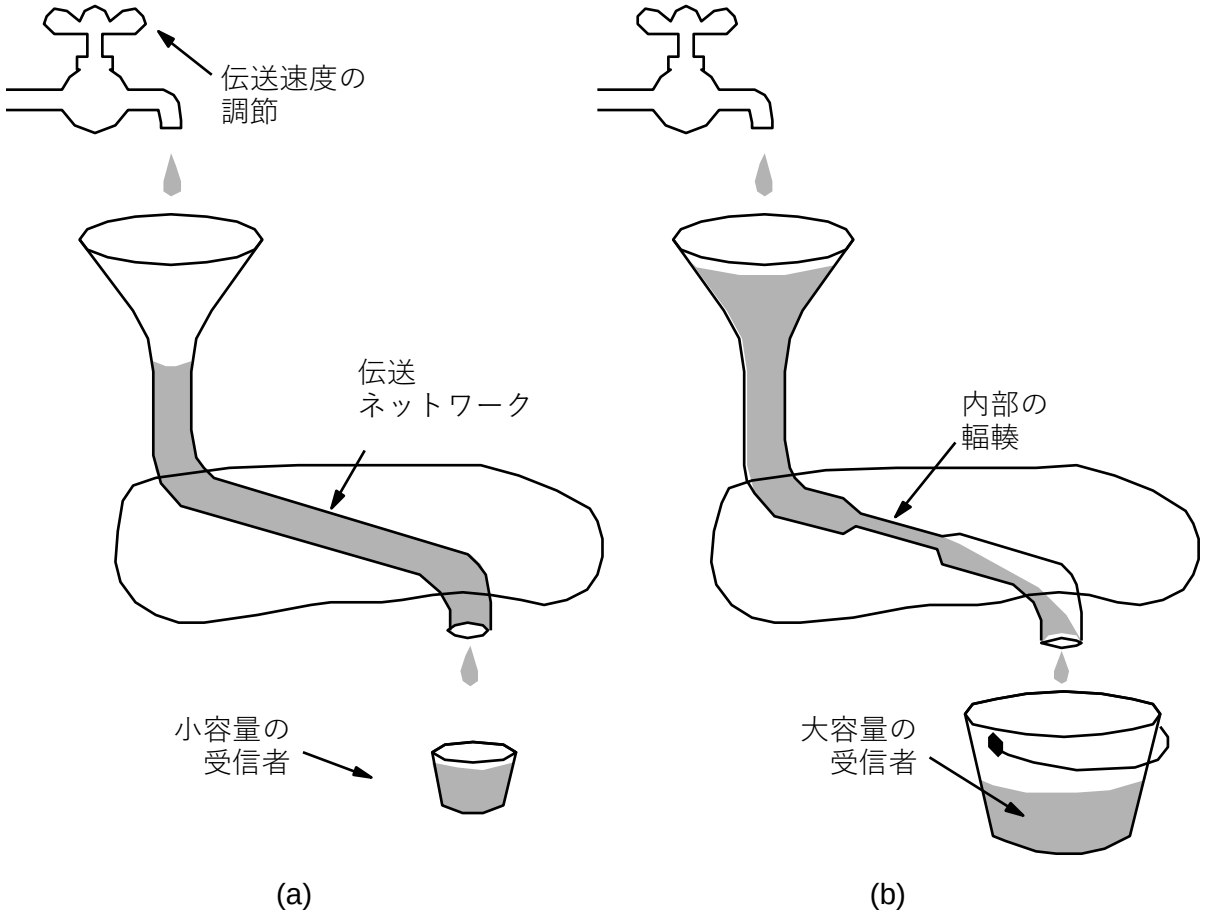
→ウィンドウサイズ: 20kbit=2500バイト

例：2Mbps, 往復200ms

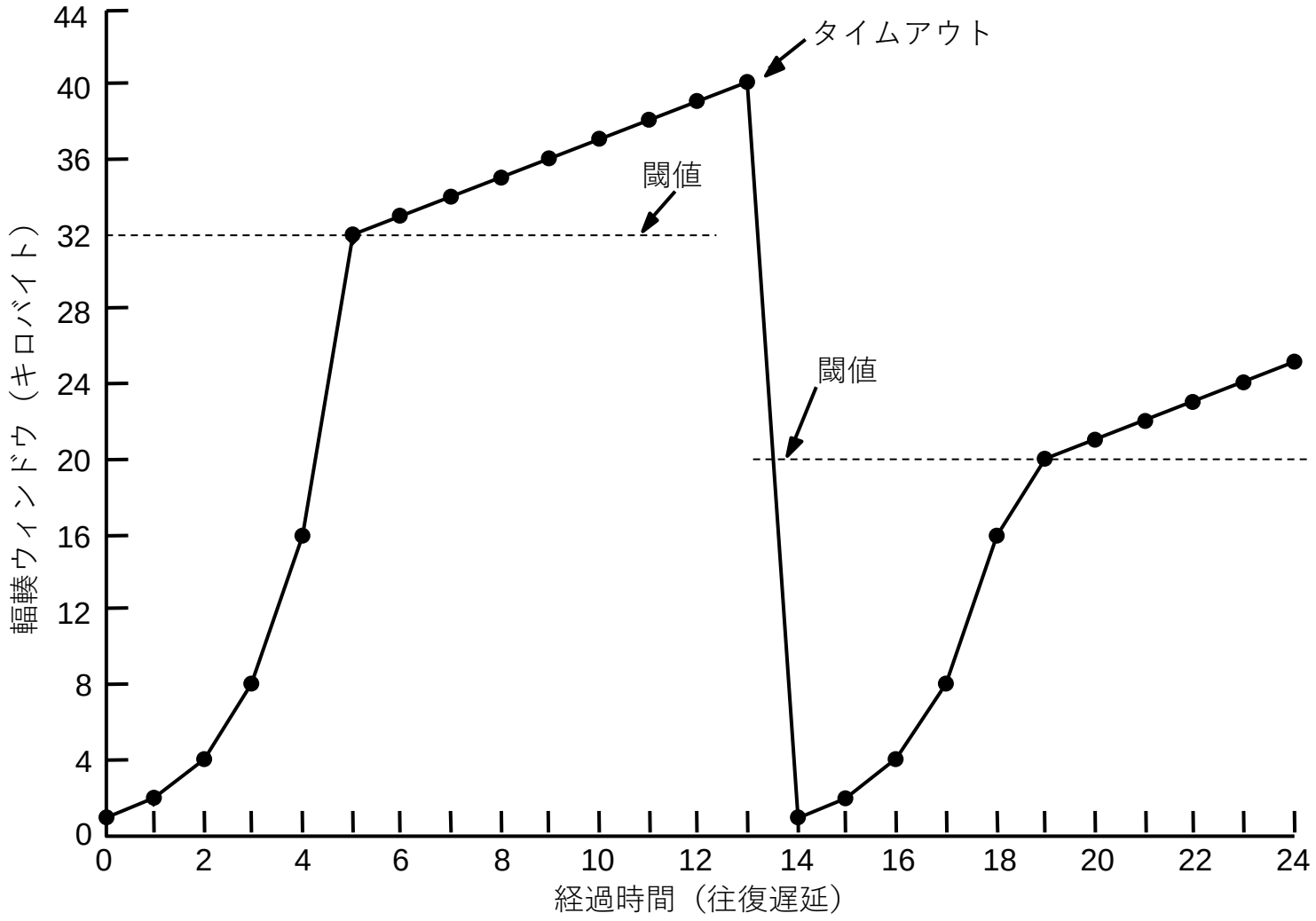
→ウィンドウサイズ: 400kbit=50,000バイト

- ウィンドウスケールオプション

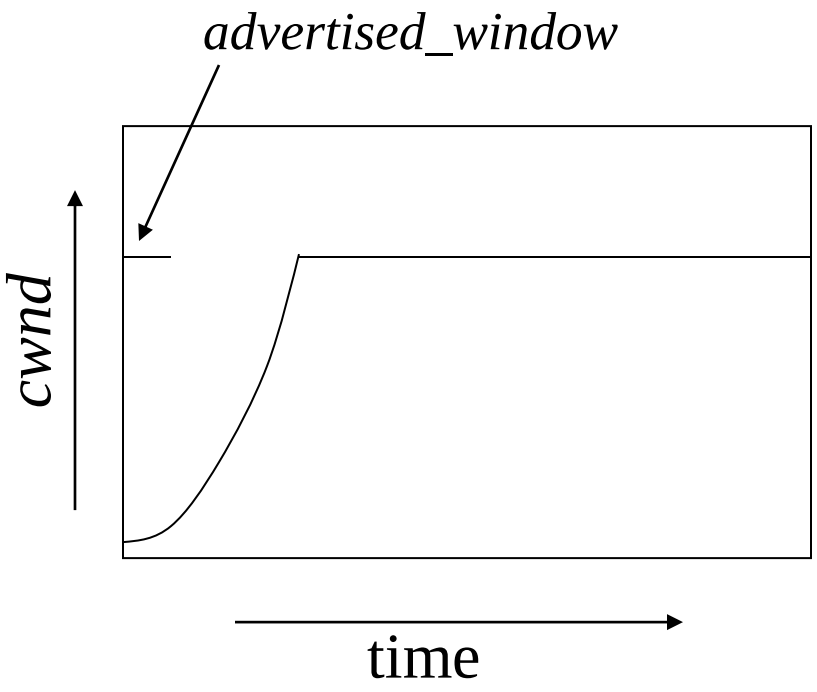
フロー制御 と 輻輳制御



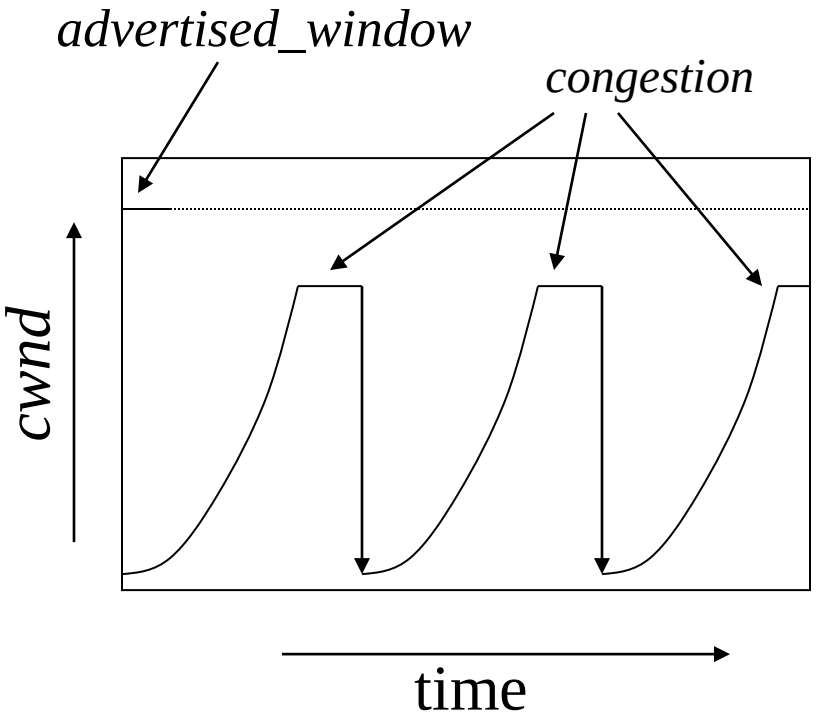
スロースタート・輻輳回避



TCP Congestion Window



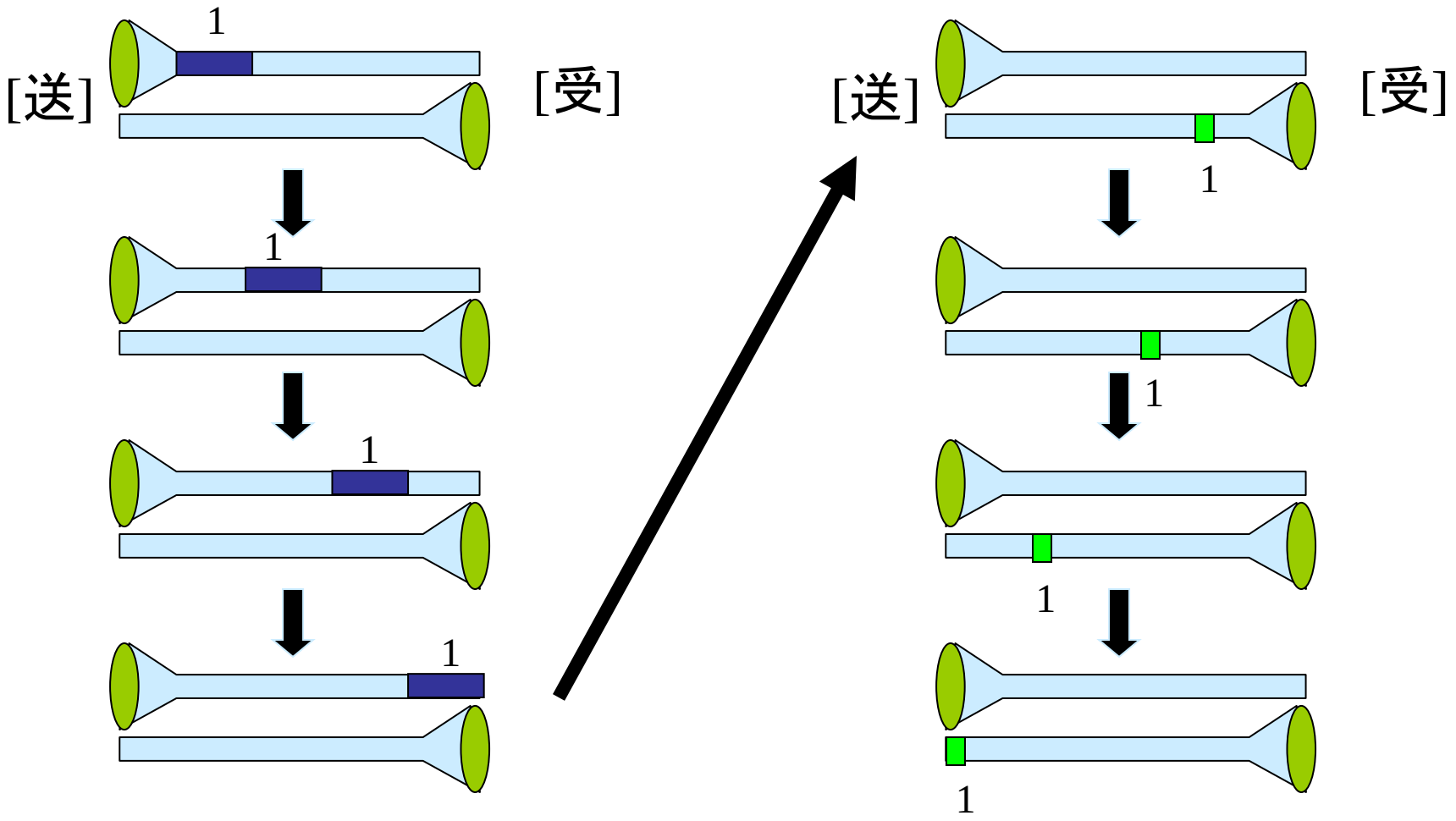
< Congestionなしの場合 >



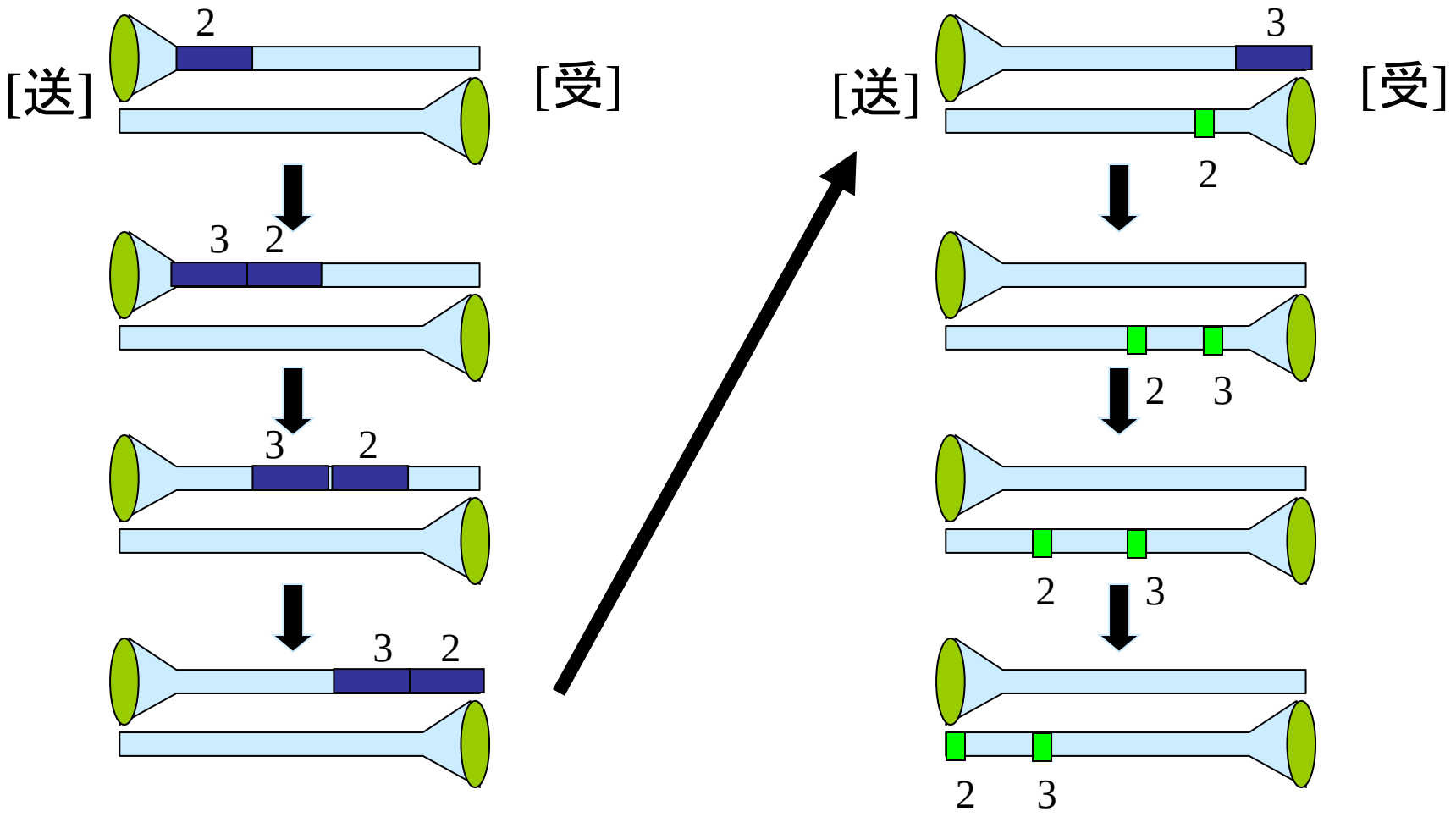
< Congestion経験の場合 >

(*) Duplicated ACKを使用せず

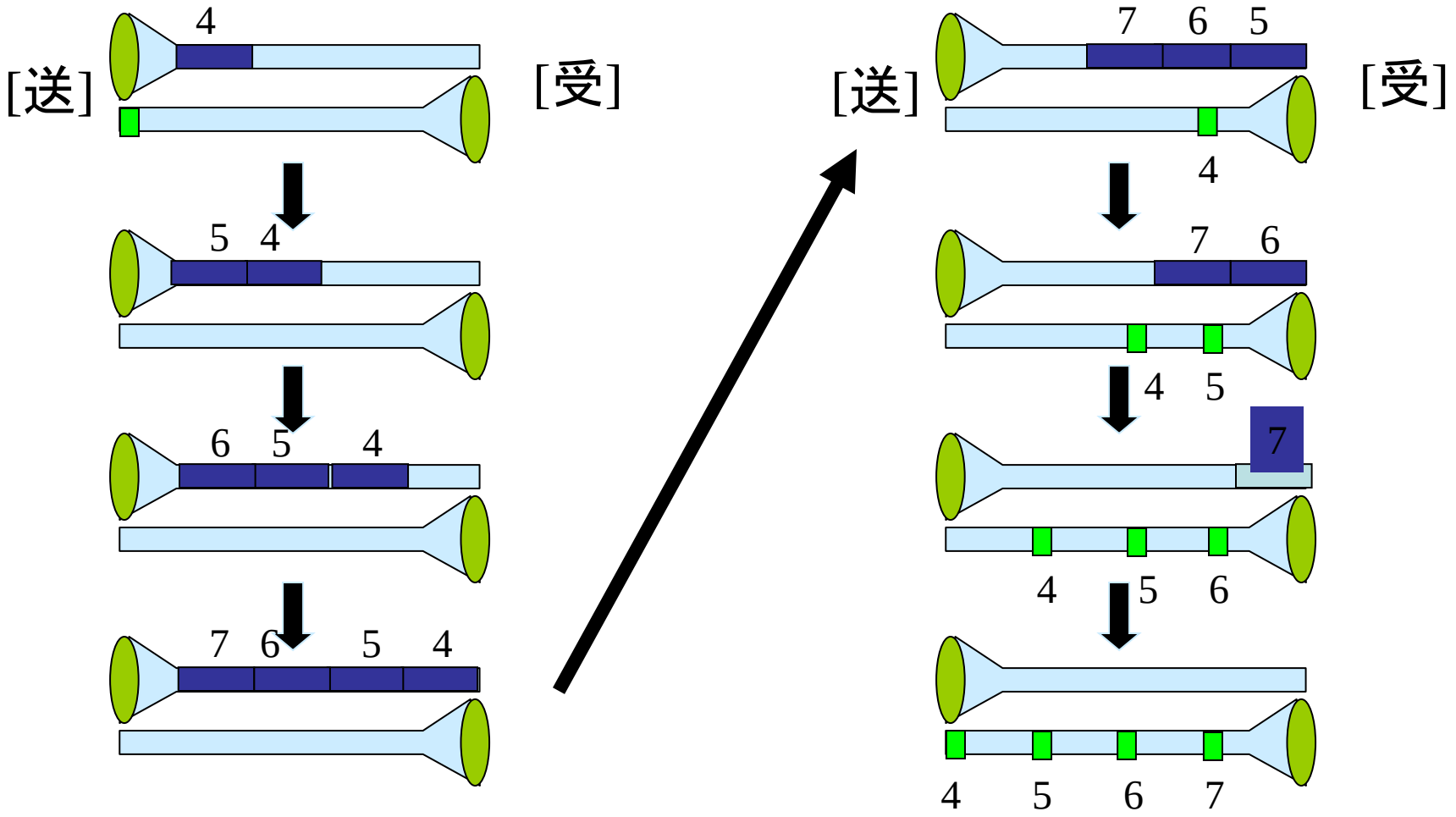
TCP Congestion Window(1)



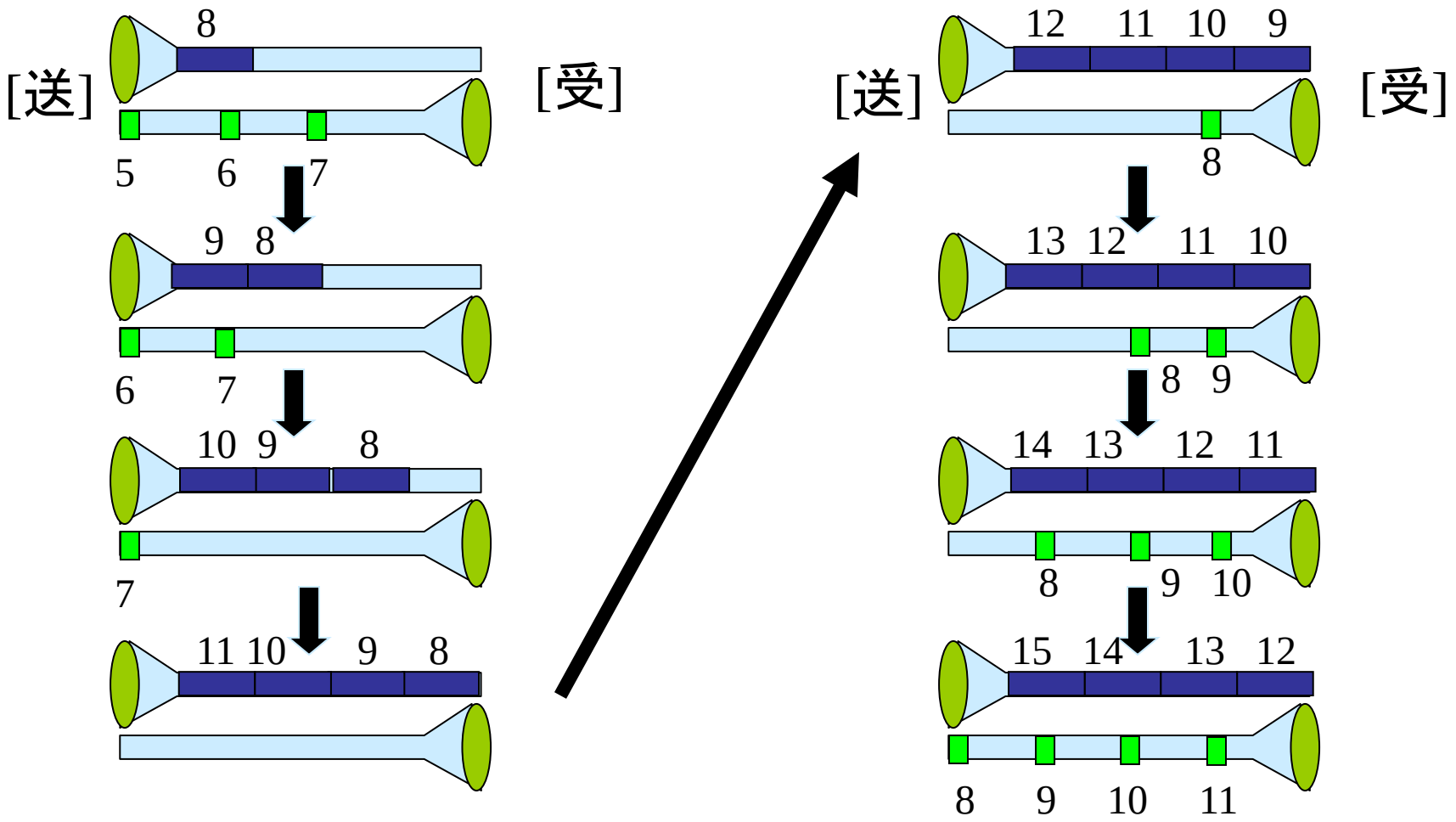
TCP Congestion Window(2)



TCP Congestion Window(3)



TCP Congestion Window(4)



必要なウィンドー幅 ≧ BWxRTT

Window Scaling for Long Fat Pipe
- RFC1323 -

Network	Bandwidth（bps）	RTT（ms）	BWxRTT（B）
Ethernet	10.000 M	3	3,750
T1（大陸間）	1.544 M	60	11,580
T1（衛星）	1,544 M	500	96,500
T3（大陸間）	45,000 M	60	337,500
OC12（大陸間）	2,400,000 M	60	7,500,000

・Max. Window Size ； 2^(16) Bytes = 64KB
→ Window Scaling ; “wscale”
wscale=n → 64 x 2^(n) windowサイズ

TCPのフロー制御

- 送信側の ウィンドー制御は、勝手に 作れる。
  - 実装は、ある程度 同じものになっている。
  - しかし、定義上、いろいろな アルゴリズムを 適用・導入可能にしてある。
- 受信側は、上限のみを 通知する。

RFC 1379 ; T/TCP
- Transaction TCP -

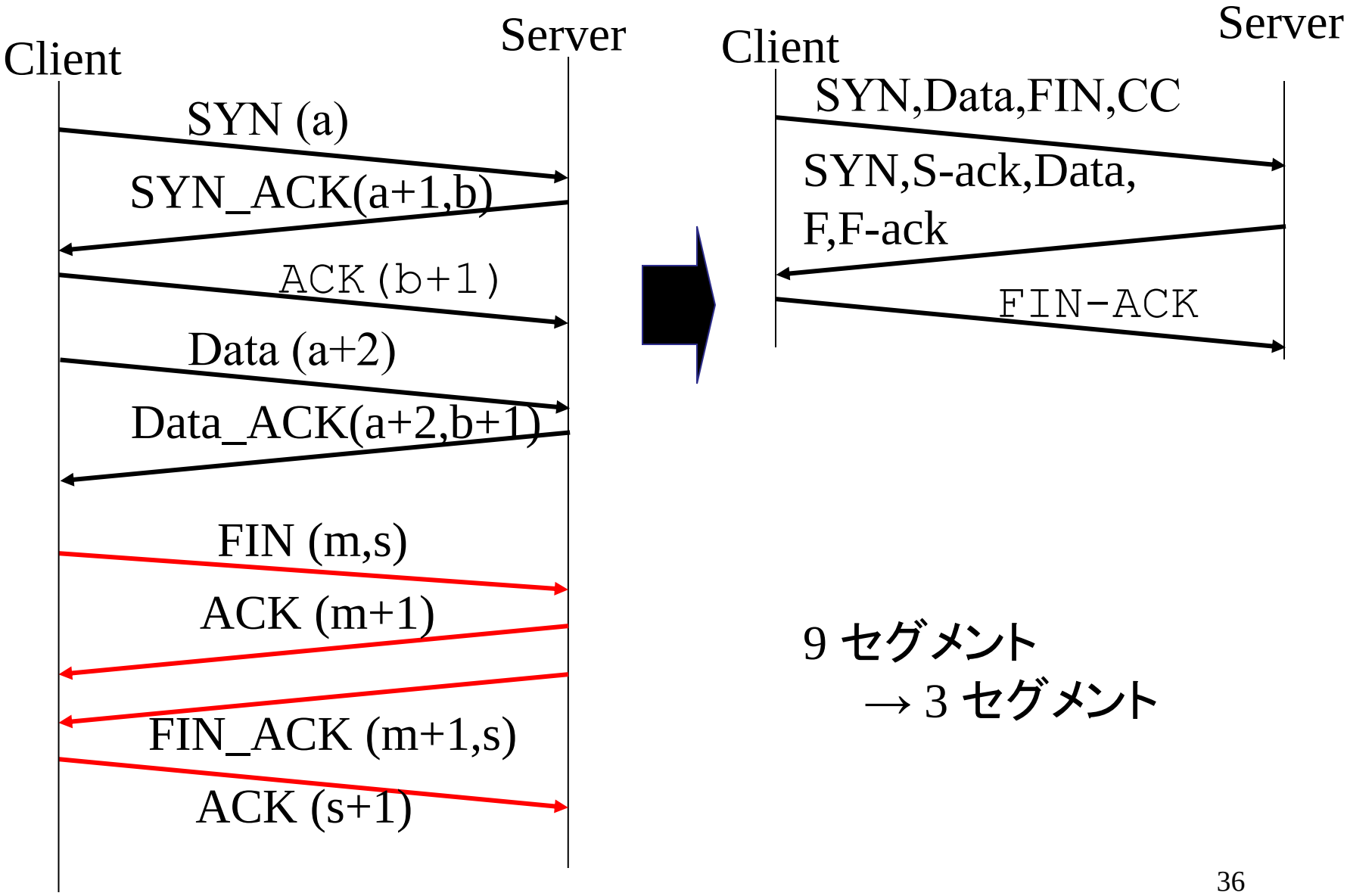
[目的]

TCPコネクションの確立・開放手続きの速度アップ

[方法]

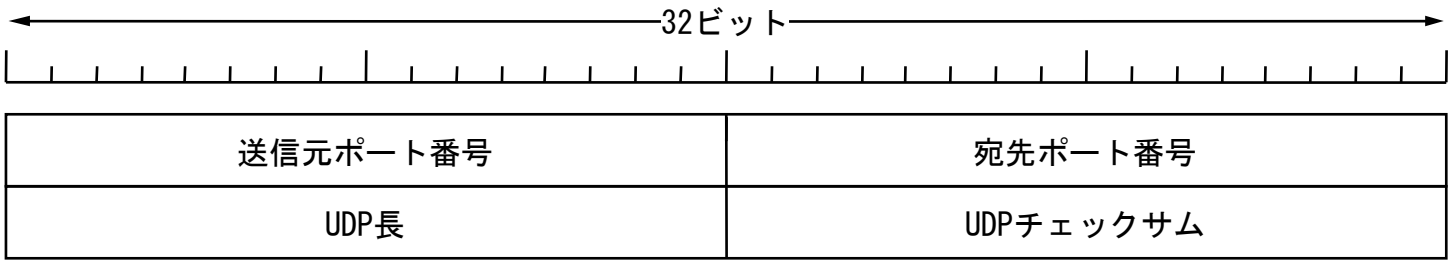
- *CC (Connection Count)* Option
- SYNへのPiggy-back ; “*half-synchronization*”
  - (1) SYN, Data, FIN, CC
  - (2) SYN, SYN-ACK, Data, FIN, FIN-ACK, CC, CC-Echo
  - (3) FIN-ACK

RFC 1379 ; T/TCP

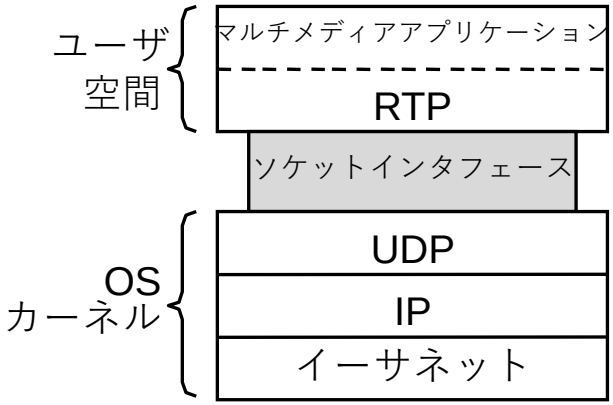


9 セグメント
→ 3 セグメント

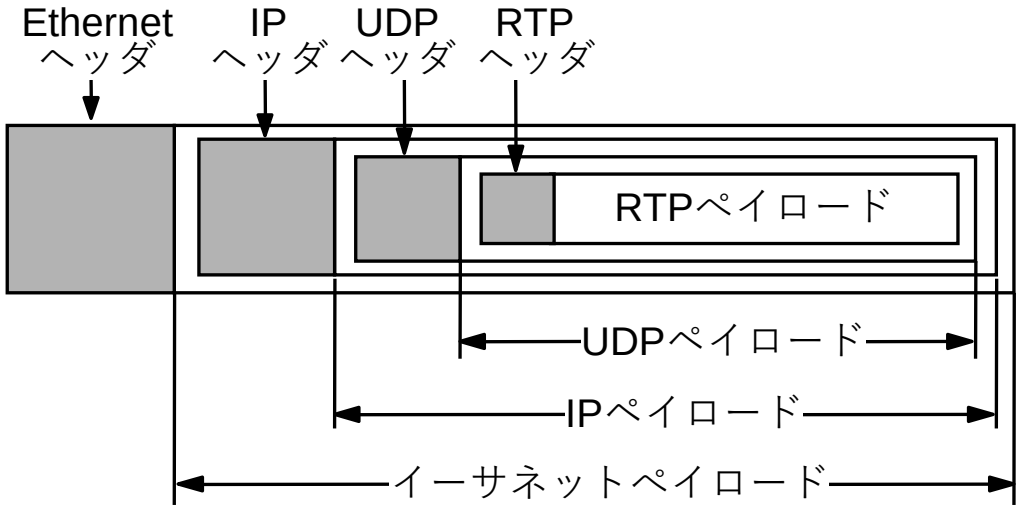
UDPヘッダ



RTP (Real-Time Transport Protocol)

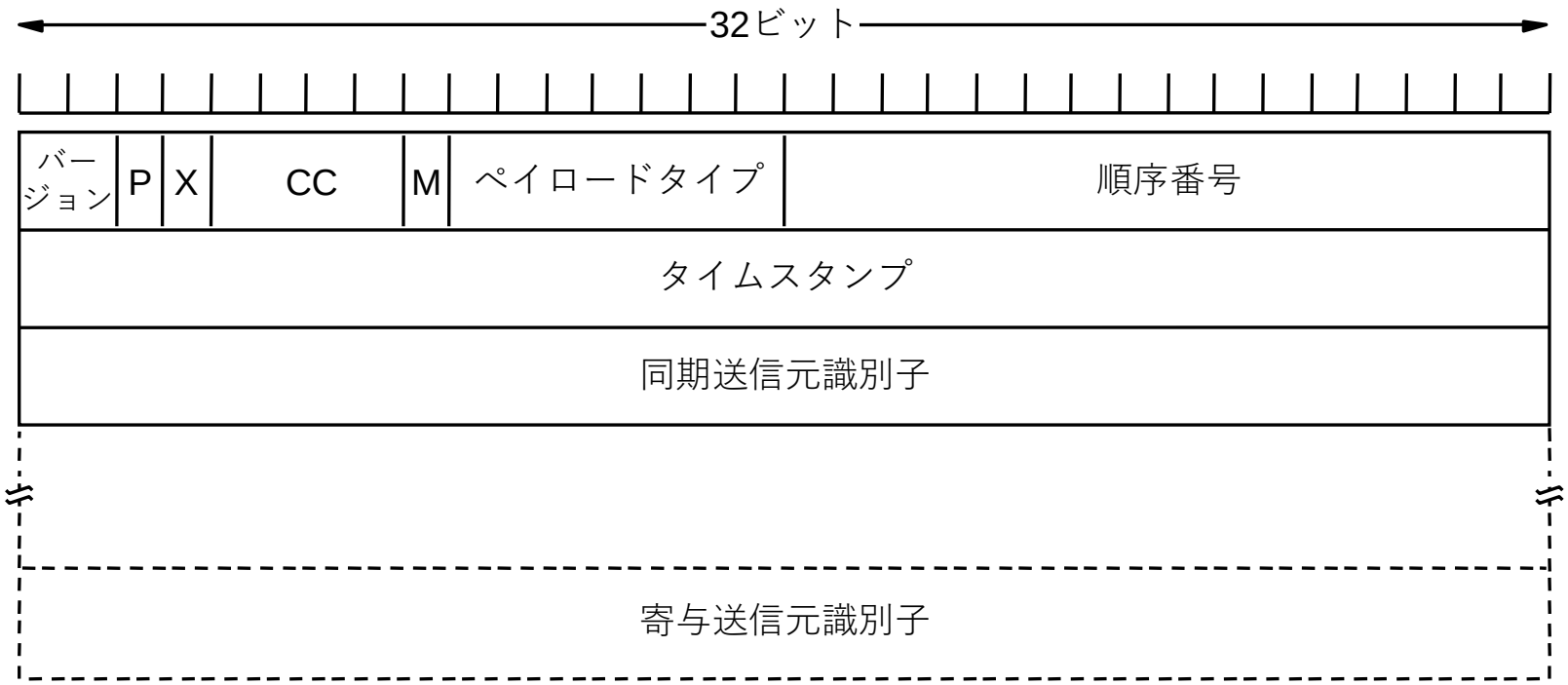


(a)



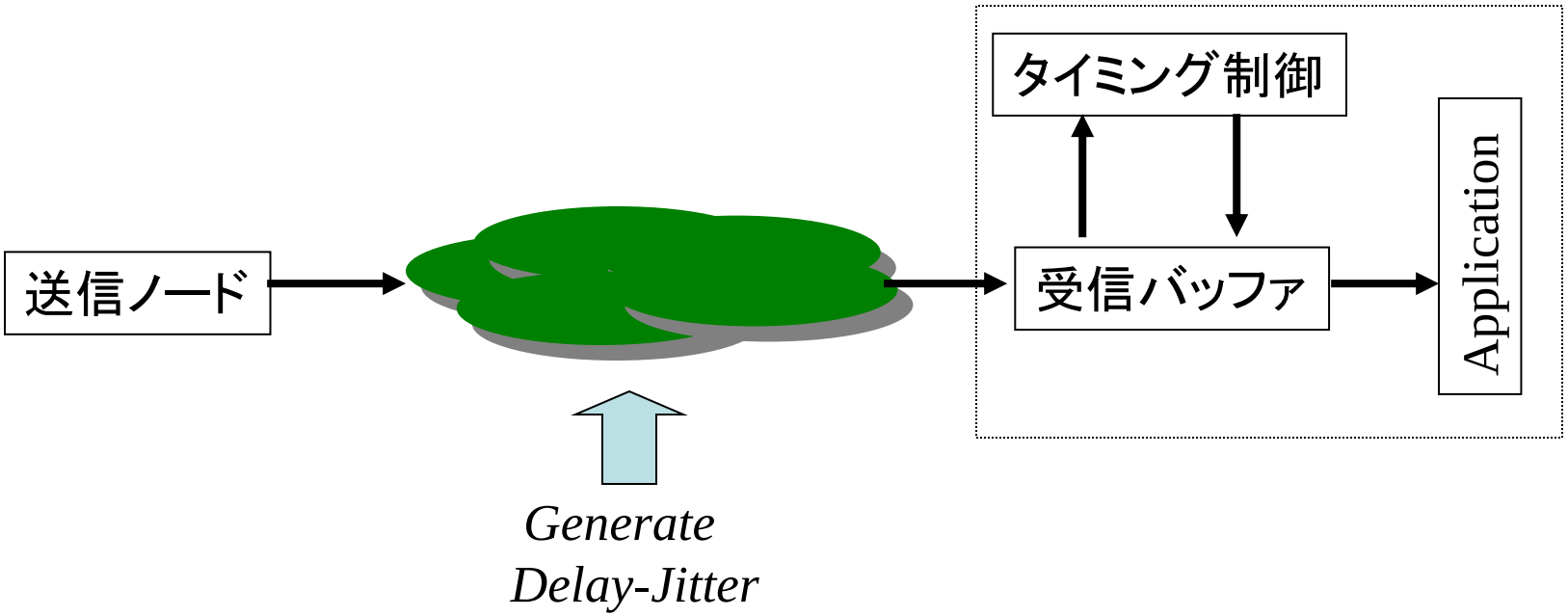
(b)

RTPヘッダ


32ビット
バージョン
P
X
CC
M
ペイロードタイプ
順序番号
タイムスタンプ
同期送信元識別子
寄与送信元識別子

RTP

- RTPの仕事;
  「受信ノードにおいて、送信側から送信される
  データの出力タイミングを再生する。」

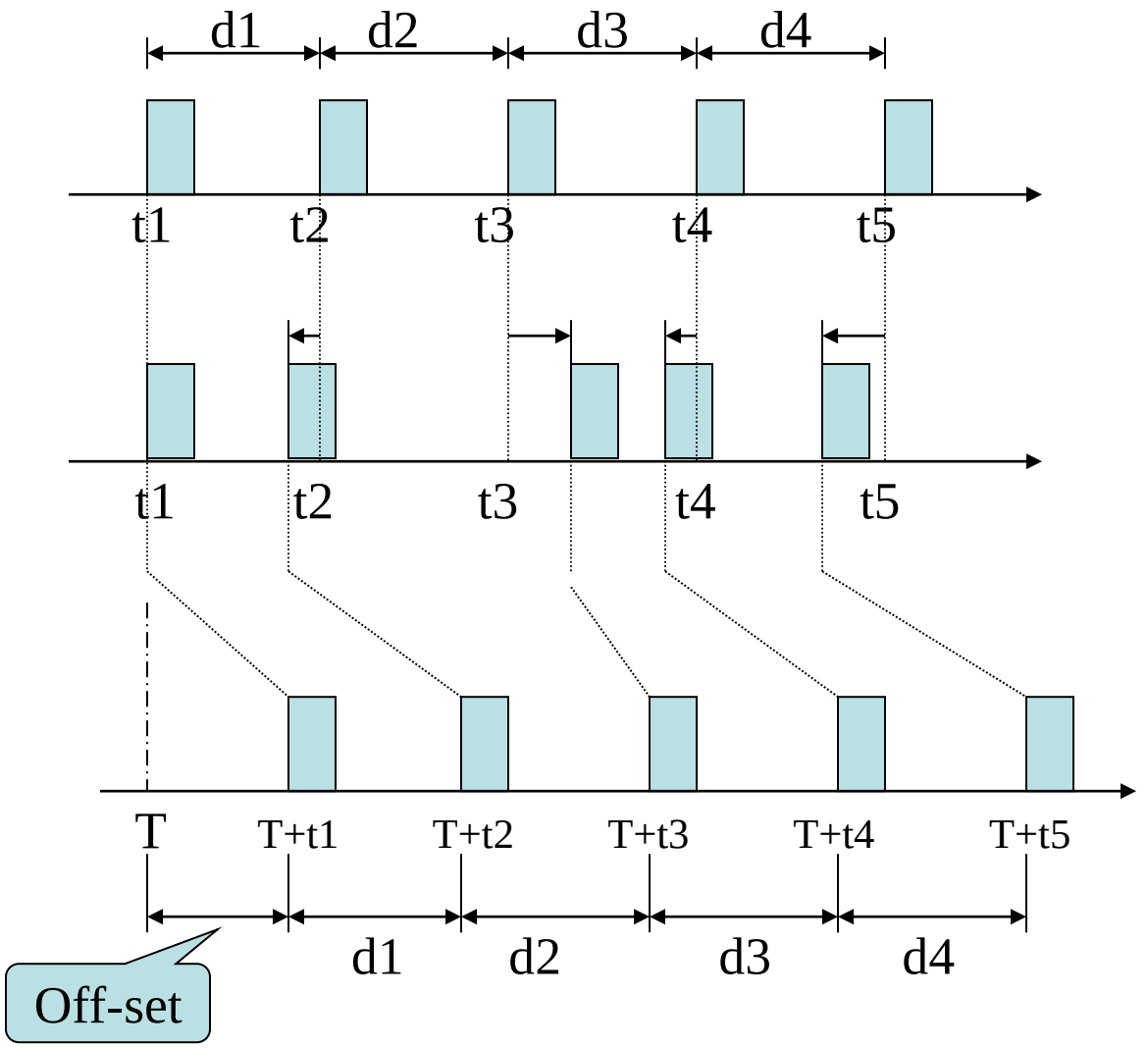


RTP

・送信側タイミング;

・受信側入力タイミング;

・受信側出力タイミング;



新しい方向性

1. マルチパス化
   – MTCP (Multi-Pah TCP)
      ➔ 複数のIPアドレスを利用
2. UDP ＋ アプリケーション での実装
   – QUIC(Quick UDP Internet Connection)
      (*) Proposed and implemented by Google
      1. 初期設定のための Hand-Shake を高速化
      2. 前方誤り訂正機能(FEC：Forward Error Correction)
      3. 暗号化
      4. 自由なフロー制御