モバイル対戦アクションゲームの通信最適化テクニック

通信遅延やパケロスへの対策をがんばる

テクニカルディレクター 石森礼二

© 2018 SQUARE ENIX CO., LTD. All Rights Reserved.

前置き

通信の遅延には、それをごまかすテクニックがありますよね。

- ユーザーの操作をそのユーザーの画面にとりあえず即反映する
  通信の結果を待たず、右キーを押したらすぐ右に移動する
- 受信した通信結果が矛盾したら、通信を正として訂正する
  実はすでに敵に吹っ飛ばされていたので右に移動はできなかったとか

こうすれば、通信の劣悪さを感じづらくなります。

ただ、通信が劣悪だと矛盾が頻発するため**“対戦”**アクションゲームとしては通信もなるべく頑張る必要があります。

※ひとりプレイや、協力プレイなら、人間有利にしておけば文句でないんですけどね

© 2018 SQUARE ENIX CO., LTD. All Rights Reserved.

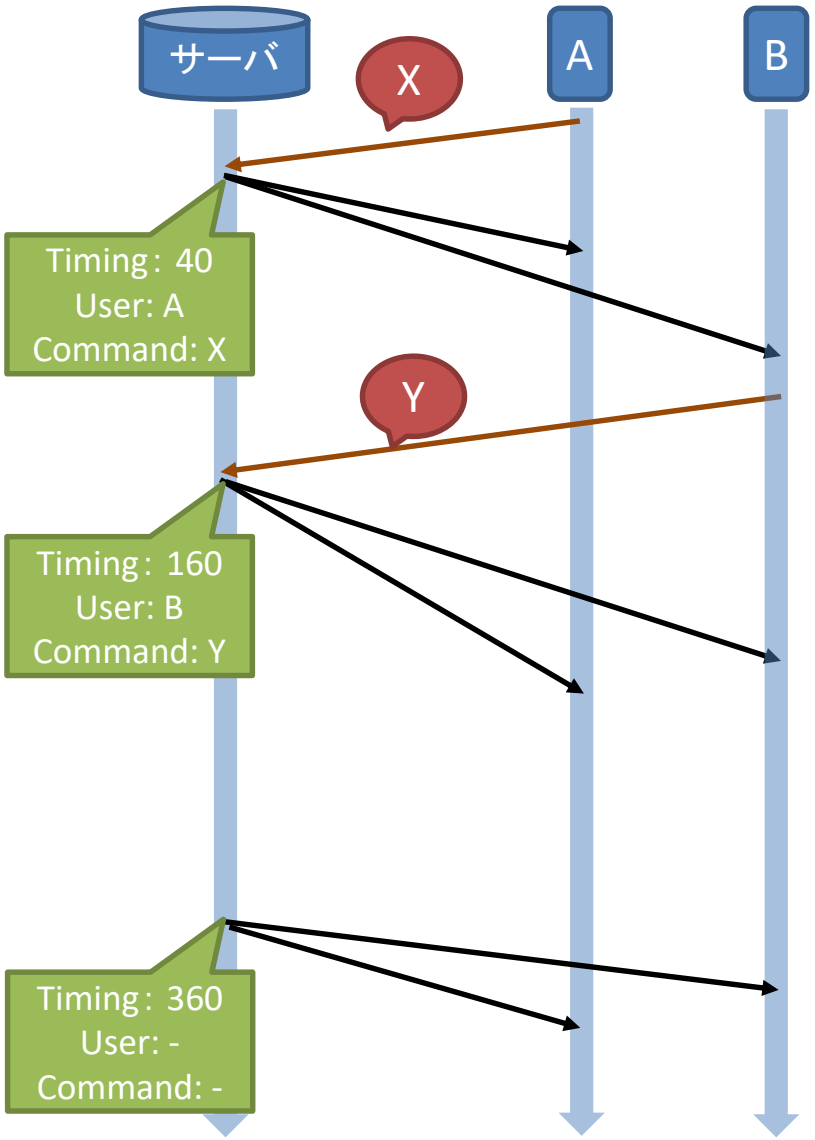
前置き

今回は通信頻度が高く、シビアな対戦アクションゲームを作るための技術を共有します。

説明のために、いわゆる**完全同期型/コマンド入力同期方式**でのゲームを想定しておきます。

© 2018 SQUARE ENIX CO., LTD. All Rights Reserved.

完全同期型/コマンド入力同期方式



- クライアントはサーバに自分が操作したコマンドを送信する
- サーバは全クライアントに、そのコマンドの実行時間を付与してブロードキャストする
- クライアントは次々届くコマンドを再生し、全クライアントで同じ画面が再現される
- パケットが届かないとクライアントがゲームを進められないので、サーバにコマンドが来なくても一定間隔でブロードキャストする

© 2018 SQUARE ENIX CO., LTD. All Rights Reserved.

共有する情報リスト

1. どのくらいの頻度で送受信をするべきか
2. RUDP(Reliable User Datagram Protocol)の恩恵と作り方
3. サーバ運用コストの下げ方

お題はこの3つです。

© 2018 SQUARE ENIX CO., LTD. All Rights Reserved.

◆どのくらいの頻度で送受信をするべきか◆

© 2018 SQUARE ENIX CO., LTD. All Rights Reserved.

通信頻度が高いとどうなる？

利点

- ゲームの更新頻度が高くなり、より正確に最新の状況を描画できる
- 端末の無線が寝なくなり、通信レイテンシが良くなる ※詳細後述

欠点

- 端末バッテリーの減りが速くなる
- サーバやクライアントの処理負荷があがる
- サーバの回線が混雑する（サーバ/クライアントモデルの場合）

ゲーム描画が60fpsだからといって、秒間60パケット送受信するのは避けたい……

© 2018 SQUARE ENIX CO., LTD. All Rights Reserved.

どのくらいの頻度でパケット送受信する？

対戦RTSのように「命令をしてから実行されるまでに、ゲーム仕様として2秒程度のラグが存在する」ゲームの場合、1秒ごと程度で良いと思います。

ですが今回のテーマは**対戦アクションゲーム**ですからそうはいきません。

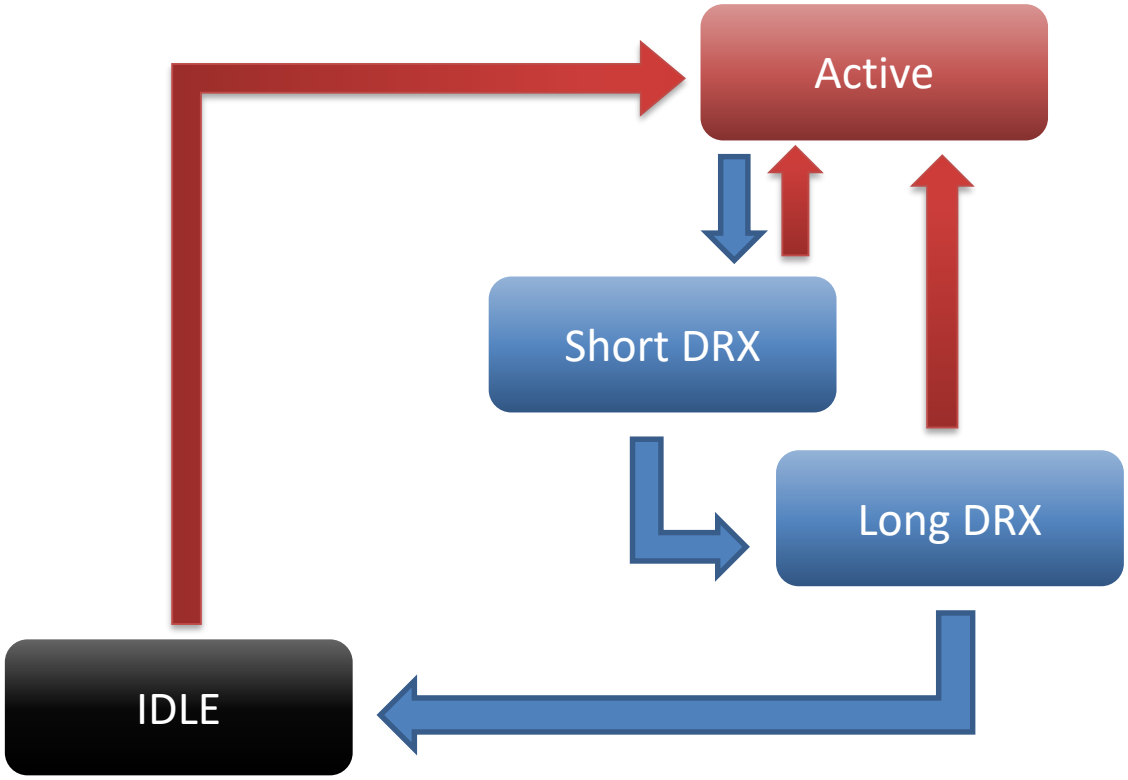
《先に結論》

100ms ～ 200ms 程度ごとに受信し続け、必要な時に送信するのがおすすめ

© 2018 SQUARE ENIX CO., LTD. All Rights Reserved.

送受信が止まると、無線が寝てしまう

LTEデバイスの状態遷移図です。

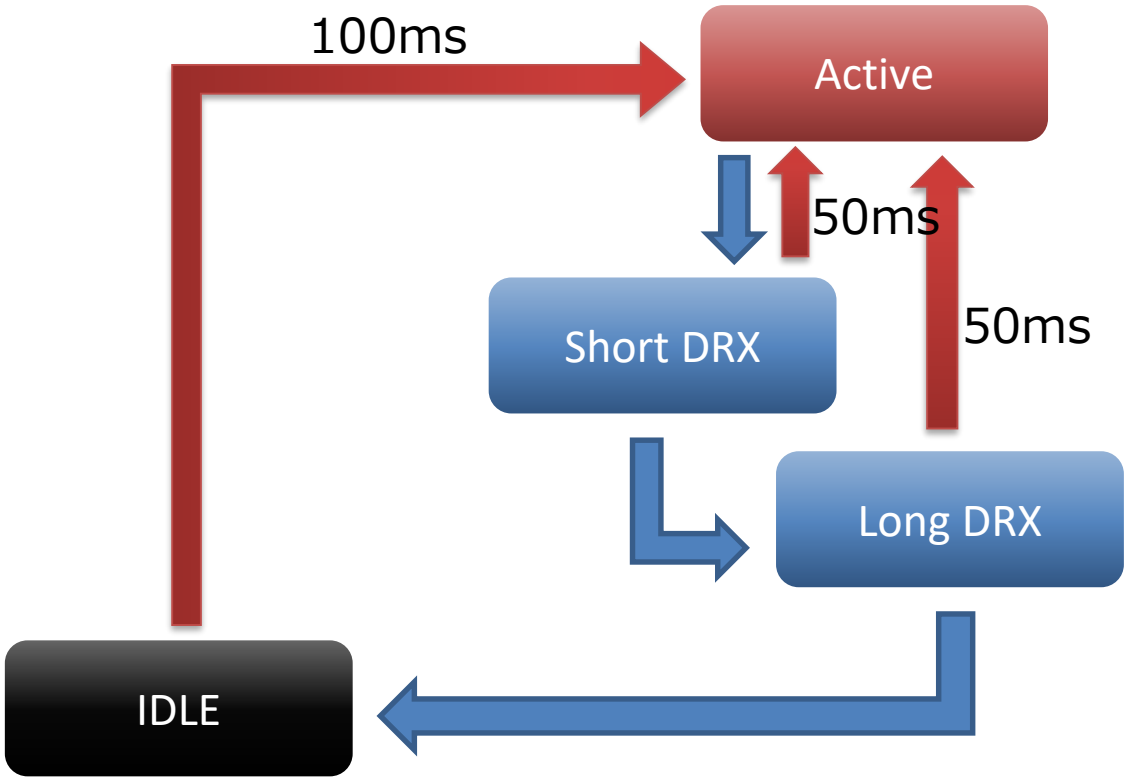


ユーザーパケットを送受信するためには、Activeである必要があります。

ですが端末は、省電力のためになるべくShort DRXに移行したがります。

© 2018 SQUARE ENIX CO., LTD. All Rights Reserved.

寝ている無線を起こすコスト

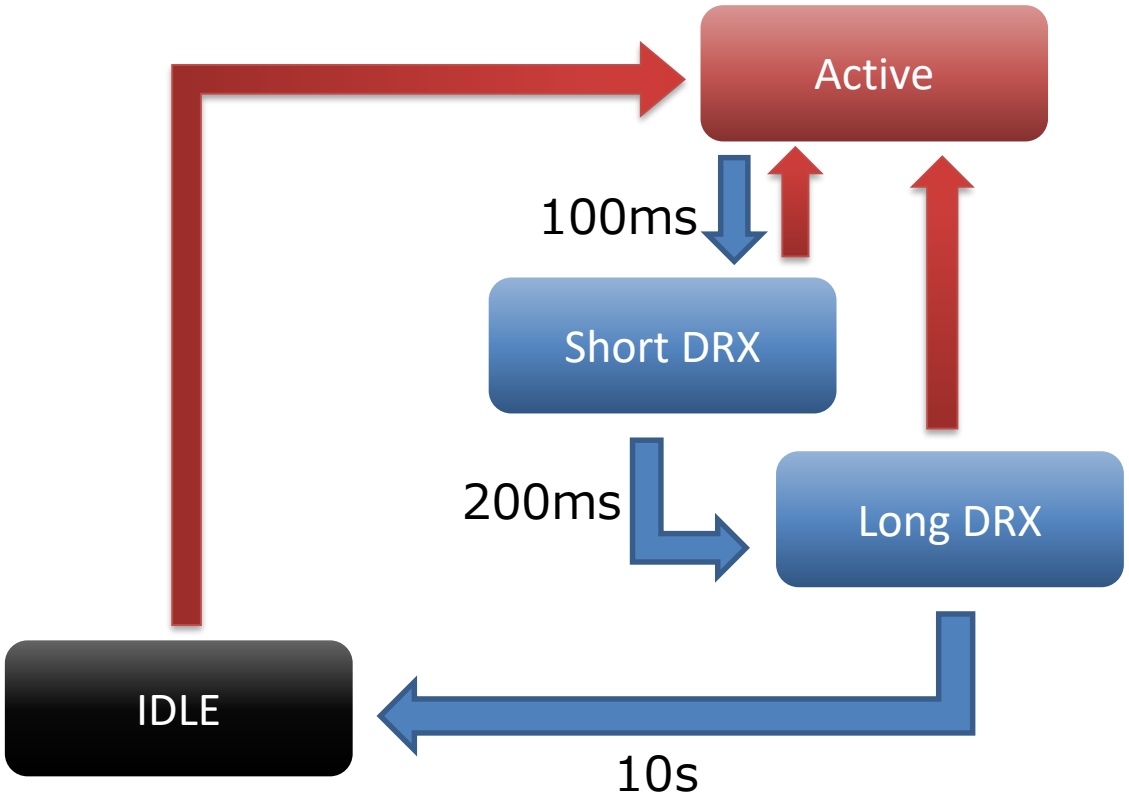


遷移に必要な時間を
表示しました。

この数字**以下**というの
が規格上の数字です。

© 2018 SQUARE ENIX CO., LTD. All Rights Reserved.

状態が遷移するタイムアウト時間



パケットが途絶えている状況で、状態が遷移するタイムアウト時間を表示しました。

参考設定値だそうですが、実際は**もっと長い**ようです。

IDLEになる条件は10秒ですから、ここの心配はなさそうです。

© 2018 SQUARE ENIX CO., LTD. All Rights Reserved.

パケットの送信RTT

100ms	200ms	300ms	400ms	500ms	600ms	700ms	800ms	900ms	1000ms
43	49	58	50	51	97	87	86	87	86
54	49	67	199	120	76	87	85	86	85
44	48	47	49	107	84	86	85	86	84
42	48	266	238	89	84	86	84	85	85
43	47	46	47	218	84	85	84	54	84
42	56	255	47	278	143	95	83	104	93
56	56	54	46	297	83	85	82	83	73
51	55	264	46	57	82	85	81	53	77

最上段は「パケットの受信間隔」です。
その状況で、時々飛ばす送信パケットのRTT(ms)を表に載せています。

※第3.9世代移動通信システムでの実験結果

© 2018 SQUARE ENIX CO., LTD. All Rights Reserved.

パケットの送信RTT

100ms	200ms	300ms	400ms	500ms	600ms	700ms	800ms	900ms	1000ms
43	49	58	50	51	97	87	86	87	86
54	49	67	199	120	76	87	85	86	85
44	48	47	49	107	84	86	85	86	84
42	48	266	238	89	84	86	84	85	85
43	47	46	47	218	84	85	84	54	84
42	56	255	47	278	143	95	83	104	93
56	56	54	46	297	83	85	82	83	73
51	55	264	46	57	82	85	81	53	77

- ～200ms間隔で受信し続けていればRTT50ms程度で安定する
- 300ms～500msのところは異常に遅くなることが頻発する
- 600ms～ならば、全体的に少し劣るが安定している

© 2018 SQUARE ENIX CO., LTD. All Rights Reserved.

RTT計測からの推測

- ～200msで受信し続けていれば常にActiveなので、スムーズに送信が可能。

- 600ms～ではShort DRXに移行しているため、Activeへ遷移するためのコスト40ms程度が上乗せされている。カタログ通り。

- 300ms～500msでは150ms以上ものコストが上乗せされている。推測するに、Short DRXへ移行中にユーザーパケットを処理しようとすると「いったんShort DRXに移行しきってからActiveに移行する」のではないか。
  Short DRXに移行する際にネットワークリソースが解放されるため、その解放処理コストが大きいのでは？

© 2018 SQUARE ENIX CO., LTD. All Rights Reserved.

RTT計測からの結論

RTSなら600ms～を選択したい。
ただ今回はアクションゲームなので～200msが妥当か。

常に無線がActiveなので、バッテリー消費量が大きいのは仕方ない。

© 2018 SQUARE ENIX CO., LTD. All Rights Reserved.

送受信間隔の下限を決める

バッテリーやサーバ負荷を気にしなければ、短いほど良い。
なるべく最新の情報をもらえた方が、ゲームは正確になる。

そのゲームに求められる更新頻度にあわせて決めるのは当然だが、指標は欲しいところ。

© 2018 SQUARE ENIX CO., LTD. All Rights Reserved.

そもそもクラウドサーバはどのくらいパケットを捌けるのか

試しに、NiftyCloudにサーバを2台借りて、その2台の間で
100万ppsずつ送信と受信をさせてみた。

※pps = packet per second
※回線ではなく、OSがどのくらいさばけるかの実験です

© 2018 SQUARE ENIX CO., LTD. All Rights Reserved.

怒られた

© 2018 SQUARE ENIX CO., LTD. All Rights Reserved.

社内のインフラ管理部門に怒られた

攻撃と勘違いされました。

※限界試験を行うとは伝えていたのですが、スループット実験だと思われてた…
※隔離された実験グループ内なので、他の利用者に迷惑はかかってないです

100万ppsは出なかったです。

40万pps超えたあたりから、OSが検知できない部分でパケットが消失するようになりました。

おそらく、vNIC側が限界を迎えたのだと思われます。

© 2018 SQUARE ENIX CO., LTD. All Rights Reserved.

クラウド側での基準は10万pps

そもそもクラウド側としては、1台あたり**10万pps**くらいに抑えて欲しいみたいです。

1台で1万人くらいはさばきたいので、100msを下限としました。

0.1秒単位で画面が更新されれば、十分でしょう。

© 2018 SQUARE ENIX CO., LTD. All Rights Reserved.

結論

クライアントは100ms ～ 200ms 程度ごとに受信し続け、必要な時に送信するのがおすすめ

ただしこの実験は第3.9世代移動通信システム(通称4G)でのものです。

第4世代通信システムでテストしたところ、無線の状態遷移コストが**10ms**くらいまで下がっているようです。

※4G+とか、PREMIUM-4Gとか、LTE-Advanceと呼ばれる2015年前後から普及し始めたやつ

100msで受信し続けている時とそうでない時の差は「受信し続けてる方が若干良いが…誤差だな」くらいになりました。

© 2018 SQUARE ENIX CO., LTD. All Rights Reserved.

◆RUDPの恩恵と作り方◆

© 2018 SQUARE ENIX CO., LTD. All Rights Reserved.

RUDP(Reliable User Datagram Protocol)を作ろう

TCP/IPと同じ、到達保証、順番保証機能を実装したUDPです。

なんでTCP/IPのソックリさんを、UDPで再実装する必要があるのでしょうか？

© 2018 SQUARE ENIX CO., LTD. All Rights Reserved.

TCP/IPはアクションゲームに向かない

TCP/IPはそもそもモバイルアクションゲームには向かない仕組みです。

- 受信パケットがロストすると、パケットが再送されて届くまでその先が受信できない
- 送信も同様
- 端末のIPが変化すると切断される
- サーバ処理負荷が高い

環境によってパケロスしやすく、Wifiとの切り替わりでIPが変化し、大量のユーザーに遊んでもらう必要がある。というF2Pモバイル対戦アクションゲームにとってはどれもキツイですね。

RUDPを作れば、これらを解決できます。

© 2018 SQUARE ENIX CO., LTD. All Rights Reserved.

パケットロストってそうそう起こるのか？

日本の都会だとなかなか実感しづらいですが、それでも22時頃に計測すると0.5%～1.0%くらいは起こるようです。

電波の悪いところや、移動中だともっとあがるでしょう。

キャリアによっても品質に差があるようです。

海外はもっと酷い。

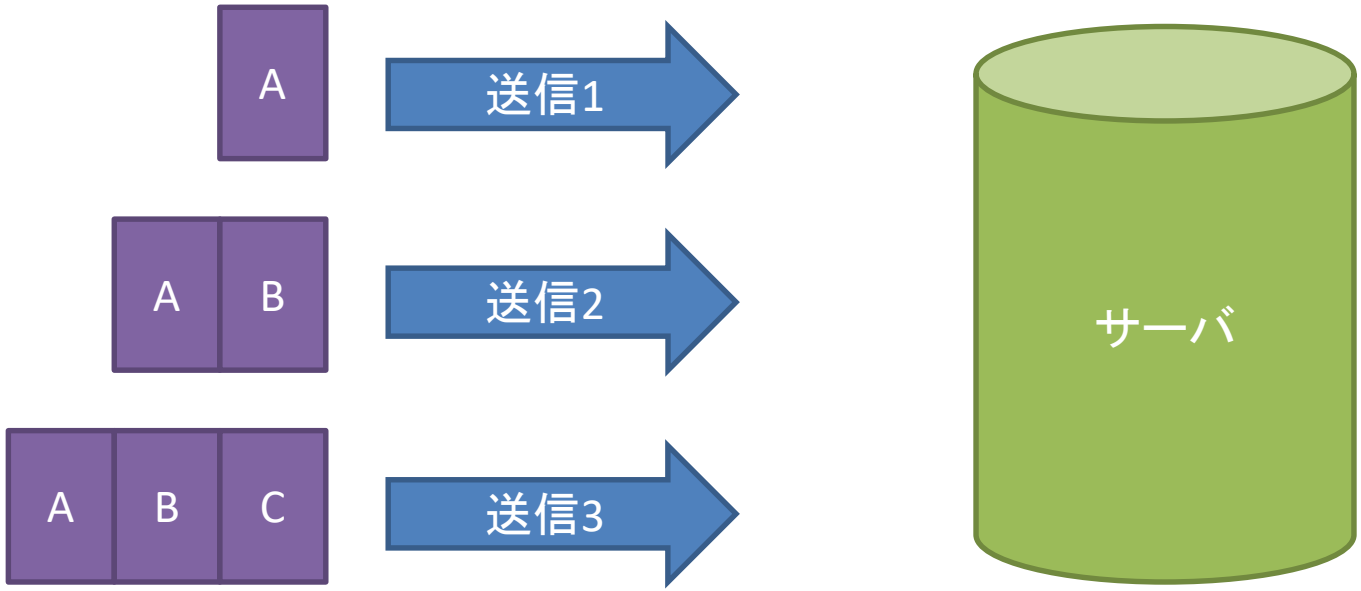
※日本、韓国は国土の狭さもあり、異常にネットワーク品質が良い

© 2018 SQUARE ENIX CO., LTD. All Rights Reserved.

RUDPの恩恵：パケロス対策

パケットに、過去数回分のパケットをくっつけて送信するようにします。

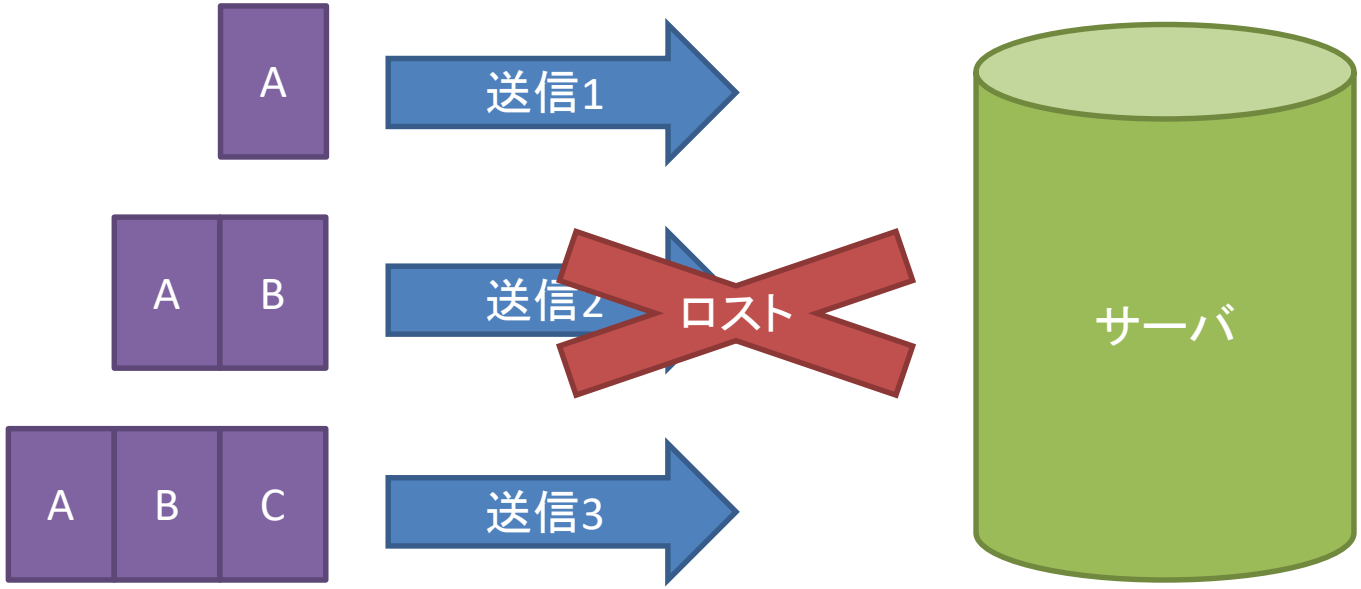
※パケットの大きさは、パケットの到達速度に影響を与えづらいです



© 2018 SQUARE ENIX CO., LTD. All Rights Reserved.

RUDPの恩恵：パケロス対策

これによって、途中のパケットがロストしても、より新しいパケットさえ届けばカバーすることができます。



© 2018 SQUARE ENIX CO., LTD. All Rights Reserved.

RUDPの恩恵：パケロス対策

高い頻度で受信をし続けるアクションゲームと相性が良い方式ですね。

※過去ログが1パケットに入らないくらい、連続でロストし続けた場合を考え、再送の仕組みはちゃんと作りましょう

© 2018 SQUARE ENIX CO., LTD. All Rights Reserved.

端末のIPが変化するとは？

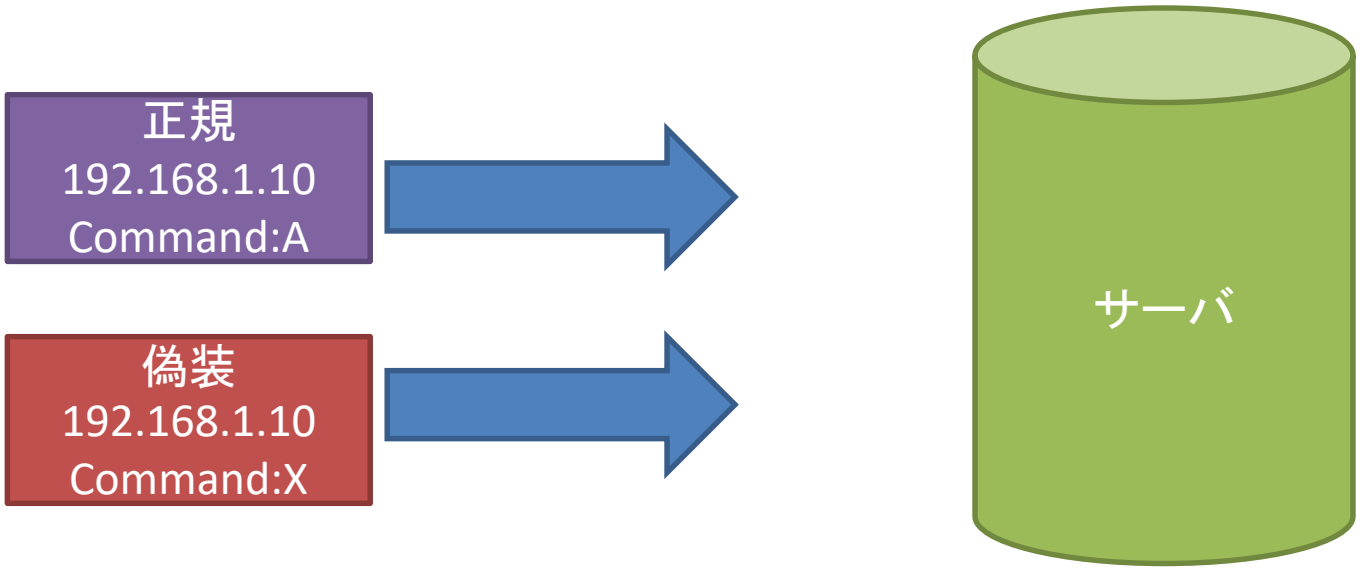
スマートフォンは、近くに利用可能なWifiがあるとそちらを利用するように回線が切り替わります。

その際に当然端末のIPアドレスが変わるので、TCP/IPだと切断されてしまいます。

© 2018 SQUARE ENIX CO., LTD. All Rights Reserved.

RUDPの恩恵：端末のIP変化対策

ところで、UDPでやりとりする場合、**送信元を偽装したなりすましパケット**に注意する必要があります。



相手のIPを盗むのは難しくない（自分のWebページを閲覧させる）し、ポート番号は65536通り全部サーバに送ってやれば良い！

© 2018 SQUARE ENIX CO., LTD. All Rights Reserved.

RUDPの恩恵：端末のIP変化対策

TCP/IPではシーケンス番号まで偽装しないと成り立たないため難しいですが、UDPはシーケンス番号が無いためIP&ポート偽装だけでなりすませます。

つまりUDPでは、**送信元IP＆ポートに頼らず**本人保証をする仕組みがそもそも必要なのです。（※後述）

よって、UDPでは**送信元IPの変化で切断する必要がなくなります。**

今回はパケットを暗号化し、「正しく復号できるパケットを送ってくれば、それは本人からのパケットである」という仕組みにします。

© 2018 SQUARE ENIX CO., LTD. All Rights Reserved.

TCP/IPのサーバ処理負荷が高いのはなぜ？

TCP/IPではrecv関数でパケットを受信するわけですが、システムコールなので呼び出し負荷が高いです。

これを大量に呼ぶ必要があるため、システムコール負荷でCPUがパンクします。

© 2018 SQUARE ENIX CO., LTD. All Rights Reserved.

RUDPの恩恵：サーバ処理負荷を大幅に下げる

UDPではrecvmmsgという関数を使うことができ、これがすさまじく速い。

詳細は後述します。

© 2018 SQUARE ENIX CO., LTD. All Rights Reserved.

ゲーム用RUDPの実装方針

- パケットを飛ばすとき、過去のパケットも一緒に入れてパケロス対策する
- 長期間パケロスし続けた場合に備えて、再送機能を実装する
- ハッシュによる改ざん防止機能と、ハッシュ部も含めての暗号化をおこない、正しくハッシュチェックまで通ったパケットを本人のものとして扱う

どうせチーター対策に通信の暗号化は必須なので、RUDPの実装に含めてしまいましょう。

© 2018 SQUARE ENIX CO., LTD. All Rights Reserved.

RUDPの実装例

```
struct HeaderRUDP {
  uint16_t session_id;
  uint8_t flg;
  uint8_t channel_no;
  uint16_t sequence_no;
  uint16_t ack_no;
  uint64_t hash;
  byte data[0];
};
```

こんな感じ。
ハッシュとデータ部を暗号化し、受信側は復号した後ハッシュが一致するかをチェックし、本人確認とする。

© 2018 SQUARE ENIX CO., LTD. All Rights Reserved.

RUDPの実装例

```
struct HeaderRUDP {
  uint16_t session_id;
  uint8_t flg;
  uint8_t channel_no;
  uint16_t sequence_no;
  uint16_t ack_no;
  uint64_t hash;
  byte data[0];
};
```

セッションID：
接続を管理するためにサーバが割り振った番号。
送信元IP&ポートではなく、この番号を接続管理に使うため、送信元IPが切り替わっても影響を受けない。

© 2018 SQUARE ENIX CO., LTD. All Rights Reserved.

RUDPの実装例

```
struct HeaderRUDP {
  uint16_t session_id;
  uint8_t flg;
  uint8_t channel_no;
  uint16_t sequence_no;
  uint16_t ack_no;
  uint64_t hash;
  byte data[0];
};
```

パケットの種類flg：
ACK番号を返すだけのパケットか、データが含まれたパケットか…。といった区別をするためのフラグ。

© 2018 SQUARE ENIX CO., LTD. All Rights Reserved.

RUDPの実装例

```
struct HeaderRUDP {
  uint16_t session_id;
  uint8_t flg;
  uint8_t channel_no;
  uint16_t sequence_no;
  uint16_t ack_no;
  uint64_t hash;
  byte data[0];
};
```

チャンネル：
仮想的に複数の接続を持たせるための概念。HTTP/2のstreamに相当。
チャットパケットが滞っただけで、アクションゲームコマンドパケットが影響を受ける危険を回避する。といった用途に使う。

© 2018 SQUARE ENIX CO., LTD. All Rights Reserved.

RUDPの実装例

```
struct HeaderRUDP {
  uint16_t session_id;
  uint8_t flg;
  uint8_t channel_no;
  uint16_t sequence_no;
  uint16_t ack_no;
  uint64_t hash;
  byte data[0];
};
```

シーケンス番号：
今回のパケットの送信番号。

© 2018 SQUARE ENIX CO., LTD. All Rights Reserved.

RUDPの実装例

```
struct HeaderRUDP {
  uint16_t session_id;
  uint8_t flg;
  uint8_t channel_no;
  uint16_t sequence_no;
  uint16_t ack_no;
  uint64_t hash;
  byte data[0];
};
```

ACK番号：
相手のパケットを何番まで受信したかを表す。
相手にそれを知らせることで、その番号以前のデータはもう送ってこなくて良いことを伝える。

© 2018 SQUARE ENIX CO., LTD. All Rights Reserved.

RUDPの実装例

- [暗号鍵, チャンネル番号, シーケンス番号]を暗号の種とします。
  そのため、まったく同じ入力を続けて送る場合でも、シーケンス番号が変化するため暗号化後パケットの内容はまったく別になります。
- シーケンス番号がオーバーフローすると**同じ種での暗号化**が行われてしまい脆弱性となります。
  そのため通算のシーケンス番号は64bitで管理しましょう。
  パケットに載せる時は16bitにしてケチってます。
  ※16bitがオーバーフローするほどロストし続けたら、そもそも切断

© 2018 SQUARE ENIX CO., LTD. All Rights Reserved.

RUDPの実装例

- データを伴わない「ACKのみパケット」ではハッシュ値が作れません。そのためデータに無意味な乱数データを入れることでハッシュ値を作りましょう。
- シーケンス番号は「データパケット」の時と「ACKのみパケット」の時で分けましょう。そうしないと、送信を行わず「ACKのみパケット」だけ送り続けている際に**同じ種での暗号化**が行われて脆弱性となります。

© 2018 SQUARE ENIX CO., LTD. All Rights Reserved.

◆サーバ運用コストの下げ方◆

© 2018 SQUARE ENIX CO., LTD. All Rights Reserved.

UDP利用によるサーバ処理負荷の軽減

《先に結論》

定価**5000円程度/月**のサーバ1台で、同時**1万接続**を**余裕で**さばける。

50,000円/月で同時接続10万人達成だ！

やったぜ！

© 2018 SQUARE ENIX CO., LTD. All Rights Reserved.

recvmmsg/sendmmsgを使おう

Linuxには**recvmmsg**という関数が用意されており、1回のシステムコールで大量のパケットを取得することができます。
1024個ずつ受け取れば、システムコール負荷が1/1024です。

同様に**sendmmsg**という関数もあります。利用しましょう。

1万接続×10ppsパケットをrecvfromで1つずつ処理していると、それだけでCPUパワーの50%以上を持っていかれるハメになります。

※ところでrecvmmsgのタイムアウトバグは直る予定ないの？ マニュアルに書いてあるから避けれるけどバグってないタイムアウト機能欲しい

© 2018 SQUARE ENIX CO., LTD. All Rights Reserved.

そのほか、細かいサーバ調整点

- **SIMDなどのCPU拡張命令を使って、暗号ルーチンを高速化しよう**
- **OSの送信バッファ、受信バッファを大きくしておこう**
  sudo sysctl -w net.core.rmem_max=8388608
  sudo sysctl -w net.core.wmem_max=8388608
- **VNICのバッファを大きくしておこう**
  sudo ethtool -G ens192 rx 4096
  sudo ethtool -G ens192 tx 4096
- **マルチスレッドではなく、シングルスレッドでのマルチプロセスをお勧め**
  マルチスレッドはバグの宝庫。
  1コア1プロセスで動作させれば十分では？
  ログ出力機能とかはさすがにマルチスレッド化するけど…
- **マルチコアならIRQチューニングとかしても良いけど、そこまでやらなくても OSがいい感じにしてくれるみたい**

※コマンド内容は環境によって異なるはずなので、あくまで一例です

© 2018 SQUARE ENIX CO., LTD. All Rights Reserved.

そのほか、雑多なキーワード

- **SYN Flood攻撃に耐えられる設計にしておこう**
  Connect要求であるSYNパケット（相当）を受け続ける攻撃。
  この段階でサーバリソースを消費しないように工夫すること。

- **改変パケットにはエラーを返さず、黙ってよう**
  エラー返却は攻撃者にとって重要なヒント。与えないようにしよう。

- **MTUを超えないように注意**
  MTUは1パケットで飛ばせるデータサイズ。
  理論上1300バイトぐらいを上限に、なるべくなら仕様上の下限である576バイトに収めましょう。

- **L1キャッシュのラインサイズを意識すると実行速度を稼げます**
  64バイト境界を意識。

© 2018 SQUARE ENIX CO., LTD. All Rights Reserved.

そのほか、雑多なキーワード

- **std::random_device は凄い重いので使わない方針で**
  10000人を捌くサーバでは重すぎた
- **仮想関数呼び出しは遅い**
- **Windows上でも動くようにしておくと、デバッグが捗るのでお勧め**
  recvmmsgが無いのでパフォーマンスは酷いですが、動作テストには十分です。
- **現在時刻の取得は重い処理なので、なるべく呼び出さない**
  再送処理や、タイムアウト処理のために現在時刻を取得したいことは多いはずです。
  ですが現在時刻の取得は重い処理なので、1サイクルの処理開始時に取得して保存し、各処理はその値を見るようにした方が良いです。

© 2018 SQUARE ENIX CO., LTD. All Rights Reserved.

ご清聴ありがとうございました

WINDOWSはマイクロソフト コーポレイションの商標または登録商標です。
その他掲載されている会社名、商品名は、各社の商標または登録商標です。

© 2018 SQUARE ENIX CO., LTD. All Rights Reserved.