Networking in the Real World

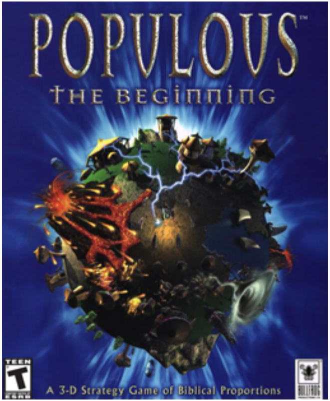
Ben Deane

3rd March 2015

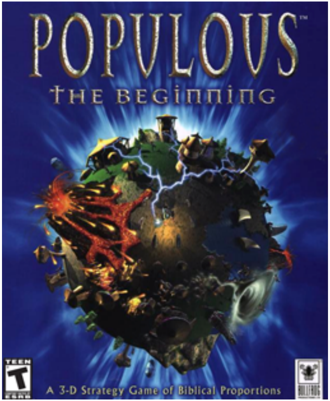
Who is this guy?

- Ben Deane
- Programmer at Blizzard on the Battle.net team
- Lifelong* network game programmer

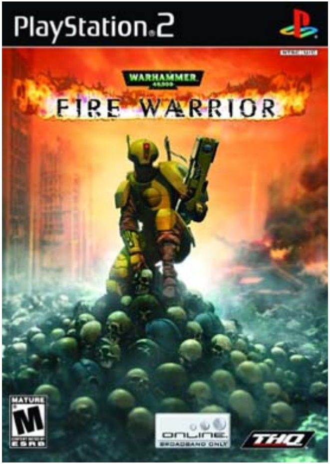
What has he done?



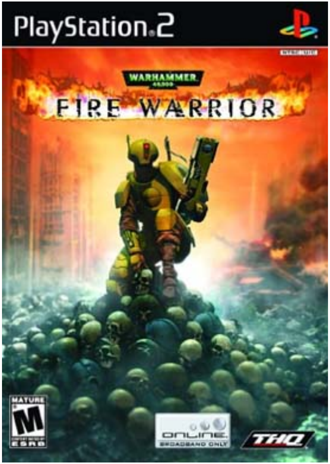
What has he done?

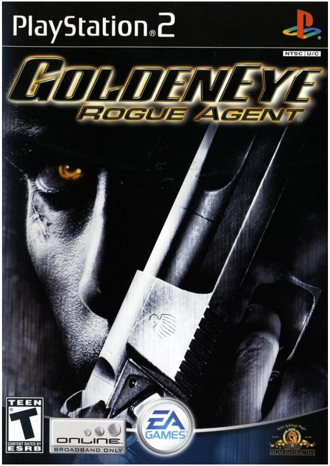


What has he done?

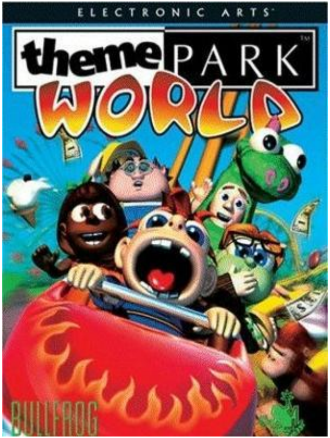


What has he done?

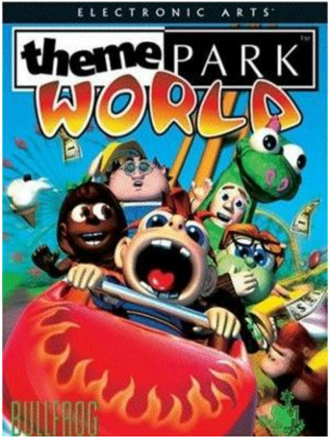


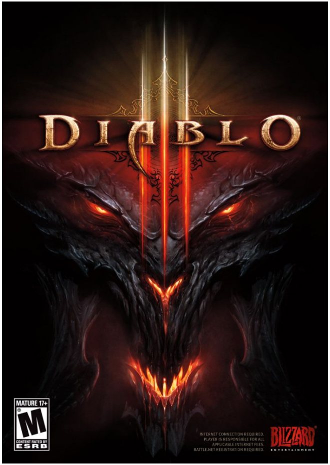


What has he done?



What has he done?





What's in this lecture?

- Real world examples
- Practical advice
- Some war stories
- Spartan slides

"Experience is simply the name we give our mistakes."

– *Oscar Wilde*

Why Network Programming?

Year	CPU (MHz)	Memory (MB)	Typical RTT (ms)
1995	90	8	300
2000	400	32	300
2005	1400	256	300
2010	2660	4096	300
2014	3330	16384	300

- Networking programming stays interesting and challenging
- Hiding latency is the constant problem to solve
- Non-network programmers just discovered concurrency?

Real World vs Academia

The Real World is what you learn but also:

- messy
- dealing with edge cases
- cutting corners
- taking advantage of hardware

TCP vs UDP

- Your most basic latency-affecting decision
- Game design and genre influences this

TCP vs UDP

TCP

- Connection, stream-oriented

UDP

- Connectionless, packet-oriented

TCP vs UDP

TCP

- Connection, stream-oriented
- 20-byte header

UDP

- Connectionless, packet-oriented
- 8 byte header

TCP vs UDP

TCP

- Connection, stream-oriented
- 20-byte header
- Guaranteed in-order

UDP

- Connectionless, packet-oriented
- 8 byte header
- Best-effort

TCP vs UDP

TCP

- Connection, stream-oriented
- 20-byte header
- Guaranteed in-order
- Nagling

UDP

- Connectionless, packet-oriented
- 8 byte header
- Best-effort
- Immediate send

TCP vs UDP

TCP

- Connection, stream-oriented
- 20-byte header
- Guaranteed in-order
- Nagling
- Socket per connection

UDP

- Connectionless, packet-oriented
- 8 byte header
- Best-effort
- Immediate send
- Single multiplexed socket

TCP or UDP?

- Your data is usually ephemeral
- It doesn't matter if one or two packets get dropped
- UDP can do NAT traversal
- UDP packet overhead is lower

Synchronizing Time I

Method 1. An NTP-like algorithm

- Estimate RTT with smoothing
- Adjust clock by (time on wire)/2
- Part of connection establishment
- Sync to epoch (eg. start of level)

Synchronizing Time II

Method 2. Iterative approach

- Client guesses time on server
- Server tells client how wrong it is
- Client adjusts its clock and repeats
- Stop when you're within tolerance

Network topologies

Peer-hosted

- single authority

"True" peer-to-peer

- distributed authority

Network topologies

Peer-hosted

- single authority
- 2x RTT

"True" peer-to-peer

- distributed authority
- 1x RTT

Network topologies

Peer-hosted

- single authority
- 2x RTT
- n-1 connections

"True" peer-to-peer

- distributed authority
- 1x RTT
- n(n-1)/2 connections

Network topologies

Peer-hosted

- single authority
- 2x RTT
- n-1 connections
- failures affect one player

"True" peer-to-peer

- distributed authority
- 1x RTT
- n(n-1)/2 connections
- failures affect everyone?

Network topologies

Peer-hosted

- single authority
- 2x RTT
- n-1 connections
- failures affect one player
- "free" consensus

"True" peer-to-peer

- distributed authority
- 1x RTT
- n(n-1)/2 connections
- failures affect everyone?
- "free" host migration

Network topologies

Peer-hosted

- single authority
- 2x RTT
- n-1 connections
- failures affect one player
- "free" consensus
- one player needs upload BW

"True" peer-to-peer

- distributed authority
- 1x RTT
- n(n-1)/2 connections
- failures affect everyone?
- "free" host migration
- everyone needs upload BW

Basic FPS Network Model

- Client-server/peer-hosted
- Time-synched to within a few ms
- Object state is transferred
- Clients converge to the true state
- 90% of data is for movement
- Semi-guaranteed protocol over UDP

Typical FPS Choices

- Two bullet types
- High fidelity human animation (=> head shots)
- Relatively few active objects at a time
- High render rate, low logic rate
- Available headless server
- Simple/Nonexistent AI

Example Semi-Guaranteed Protocol

- Entity-component model
  - Movement/Position/Rotation
  - Animation state
  - Health/Armour/Death state
- Components are marked dirty as their state is updated
- Components map to network "channels"
- Network channels are given priorities

Constructing Packets

- Keep dirty components in a priority queue
- Periodically fill a packet by priority
- Max packet size = 548 bytes
- Anything left out gets increased priority

ACKing and NAKing

- Each packet contains a sequence number
- When components are serialised they remember the sequence number
- Each packet header includes ACKs for previous packets received
  - a sequence number and a bitfield of previous acks
  - handle sequence number wraparound
- Any gaps in the ACK stream are implicitly NAKed
- Components from NAKed packets have their data re-dirtied

Compressing data

- Conserving bandwidth is important
- Bitpacking protocols are common
- Range data types
- Floating point types can be truncated
- Or quantize position in level
- 4x4 matrices are wasteful
- Rotations can be heavily quantized

Other issues

- Some things need in-order delivery
- Object creation/destruction events
- Some objects can do parallel simulation
- Others must be kept up-to-date

Race conditions

Bob's machine

Alice's machine

Race conditions

Bob's machine

Alice's machine

- Bob has 10% health.

Race conditions

Alice's machine

- Bob has 10% health.
- Alice hits Bob for 20% damage.

Bob's machine

Race conditions

Alice's machine

- Bob has 10% health.
- Alice hits Bob for 20% damage.
- Bob dies.

Bob's machine

Race conditions

Alice's machine

- Bob has 10% health.
- Alice hits Bob for 20% damage.
- Bob dies.

Bob's machine

- Bob has 10% health.

Race conditions

Alice's machine

- Bob has 10% health.
- Alice hits Bob for 20% damage.
- Bob dies.

Bob's machine

- Bob has 10% health.
- Bob picks up a health pack for a 50% health boost.

Race conditions

Alice's machine

- Bob has 10% health.
- Alice hits Bob for 20% damage.
- Bob dies.

Bob's machine

- Bob has 10% health.
- Bob picks up a health pack for a 50% health boost.
- Alice hits Bob for 20% damage.

Race conditions

Alice's machine

- Bob has 10% health.
- Alice hits Bob for 20% damage.
- Bob dies.

Bob's machine

- Bob has 10% health.
- Bob picks up a health pack for a 50% health boost.
- Alice hits Bob for 20% damage.
- Bob has 40% health.

Race conditions

Alice's machine

- Bob has 10% health.
- Alice hits Bob for 20% damage.
- Bob dies.

Bob's machine

- Bob has 10% health.
- Bob picks up a health pack for a 50% health boost.
- Alice hits Bob for 20% damage.
- Bob has 40% health.

What to do about this?

Race conditions

- Some things are problematic for races
  - eg. Health/Death
  - Divergent simulations would be bad
- You can use an accumulator model
- Take care to deal with overflow

Latency Hiding: Simple Stuff

- Clients can do simple display feedback
  - Hit animations
  - Audio
  - Blood splats
- Some things aren't going to fail
  - eg. Decrementing ammo

Predict the future

Interpolation/Prediction

Predict the future

OR (and?)

Interpolation/Prediction

Predict the future

OR (and?)

Interpolate the past

Interpolation

- Simple lerps
- Failure modes
  - Players stop
  - Warping forwards
- Take corners close
- Fundamentally a graphical/display approach

Prediction I

- Dead reckoning
- Position/Velocity/Angle
    - Acceleration
    - Rotational velocity
- Failure modes
    - Players run into walls
    - Warping back
- Take corners wide
- Fundamentally a game state/logic approach

Prediction II

- Client must reconcile its position with the server position
- Server position is in the past
- Client must rewind a little and replay recent input
- Mostly this results in seamless fixup

Prediction III

- A client can predict itself…
- Use this information to know its actions are causing divergence
- Therefore when to send an update
- You can mix a timeout with this also

Subsystem Considerations

- Play nice with the physics engine
  - Moving things into each other is a bad idea, you're not going to have a good day
  - A capped timestep is essential for your debugging sanity
  - A continuous collision system is usually necessary
- Animation tricks
  - A headless server need not pose characters until necessary

More on Update Logic

- Variable update frequency
  - Proximity
  - Velocity
  - Role (eg. target/team)
  - Visibility (PVS)

Parallel Simulation

- Some games (eg RTS) have too many objects to sync
- Input passing
- Parallel simulation

Parallel Simulation Problems

- Random events
- Camera-dependent events
- Floating point machine differences

E-sports and Fairness

- Lockstep model is old but still important
- Fairness trumps latency hiding
- High level RTS gameplay is twitch gameplay

Bug Story

Populous: The Beginning Network Model

Server

```
while (!game_over) {
  recv_client_inputs();
  send_gameturn();
  simulate();
}
```

Client

```
while (!game_over) {
  if (receive_gameturn()) {
    simulate();
  }
  render();
  send_input();
}
```

Spot the bug!

Bug Story

Populous: The Beginning Network Model

Server

```
while (!game_over) {
  recv_client_inputs();
  send_gameturn();
  simulate();
}
```

Client

```
while (!game_over) {
  while (receive_gameturn()) {
    simulate();
  }
  render();
  send_input();
}
```

`if` changes to `while`

Bug Story II

Goldeneye: Rogue Agent

Bug Story III

Firewarrior NAT negotiation

Thanks

The Real World: like Academia, except with smoke & mirrors & cutting corners & messy stuff.

Thanks for listening

`bdeane@blizzard.com`

Slides & notes available at

`http://github.com/elbeno/networking-in-the-real-world`