Week 6 – Music Control and Networks

Roger B. Dannenberg
Professor of Computer Science and Art & Music
Carnegie Mellon University

Introduction

- OSC
- Remote Music Control Protocol
- Clock Synchronization
- O2
- Network Music

OPEN SOUND CONTROL

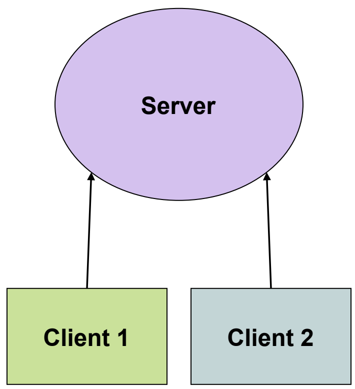
3

OSC – Open Sound Control

- Client/Server Architecture
- UDP and TCP
- Name Space
- Address Patterns
- Bundles and Atomicity
- Timestamps
- Applications
- Pros and Cons

4

Client/Server Architectures



- Client initializes contact
- Server waits on socket:
  - General server socket
  - Per-client socket
- Frequently remote procedure call based
  - Client issues call
  - Server executes function
  - Return results to client
- Basis for web servers
  - HTTP is a client/server protocol

Server Implementation Sketch

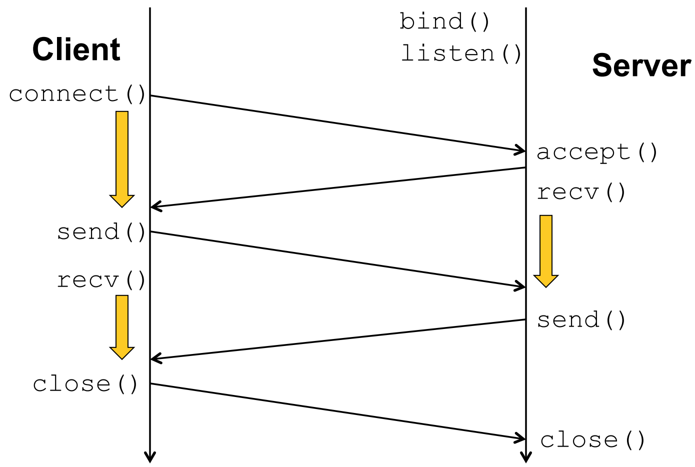
```
sad.sin_family = AF_INET; // family = Internet
sad.sin_addr.s_addr = INADDR_ANY; // IP address
sad.sin_port = htons((u_short)portno); // port #
sd = socket(PF_INET, SOCK_STREAM, ptrp->p_proto);
bind(sd, (struct sockaddr *) &sad, sizeof(sad))
listen(sd, 5)
sd2 = accept(sd, (sockaddr_ptr) &cad, &alen);
```

```
sd = socket(PF_INET, SOCK_STREAM, TCP);
connect(sd, (struct sockaddr *) &sad, sizeof(sad))
```

```
n = recv(socket, buf, len, 0);
n = send(socket, buf, len, 0);
```

```
close(socket);
```

Connection Protocol



UDP vs TCP

- UDP – "User Datagram Protocol", which you might think of as "unreliable data protocol"
  - Unreliable because no guarantees on delivery
  - Data in packets smaller than some limit
  - Order is not guaranteed either
  - Typical use on (wired) LAN very reliable
- TCP – "Transmission Control Protocol"
  - Byte stream model
  - Data *eventually* reaches destination (in order)
  - Retained data, Ack msgs, Retransmission
  - Default setting will accumulate bytes into large packets

Carnegie Mellon University
ⓒ 2019 by Roger B. Dannenberg

What can go wrong with UDP?

- Packets can be dropped
- Long messages are split across packets, so all packets have to arrive and be reassembled.
  - So usually, UDP systems send short messages that are guaranteed to fit in one packet.
  - What's a safe size? It surprising how many answers you can find to such a fundamental question. The answer seems to be around 500 bytes for the Internet, and around 1500 bytes for local Ethernet.

What can go wrong with TCP?

- TCP sends an unlimited byte stream.
  - You must delineate messages, typically prefixing a length count.
  - TCP typically delays small writes in hopes of filling a packet with additional data to achieve better throughput (more bytes/second)
    - You can send immediately by setting TCP_NODELAY option
- When a packet is lost or dropped, nothing more gets through until the sender discovers the loss and retransmits.
  - Thus, TCP stream can temporarily halt and wait, creating a substantial latency.
    - For isolated messages, transmitter fails to get an acknowledgement after a timeout period of several seconds and retransmits.
    - For frequent messages, receiver quickly detects loss by noticing an out-of-order packet (they have sequence numbers), but there's still a round-trip delay to request retransmission.

OSC Messages

- Address Pattern
  - /voice/3/freq
- Type Tag String
- Arguments
- Data Types:
  - ASCII strings
  - 32-bit float
  - 32-bit int
  - "BLOB"
  - RGB color
  - 64-bit numbers
  - Booleans
  - … and more

Name Space

- Tree-structured
- Structure defined by server
  - (not by a standard as in MIDI)
  - Is this good or bad?
- String names for nodes
  - Note that strings are globally known and available at compile time
- URL-like path names from root to message target

Address Patterns

- May contain pattern syntax:
  - * – matches zero or more characters
  - ? – matches any single character
  - [characters] – matches characters
    - Minus, e.g. [1-3] matches range of characters
    - Leading !, e.g. [!0-9] negates the match
  - {string1,string2,string3} – match a string in list
- If more than one destination matches address pattern:
  - Send copy of message arguments to each node
  - Fanout to unknown destinations
  - For example: control all "voices" with volume pedal

Bundles and Atomicity

- Bundles are sequences of messages
- All messages in a bundle are delivered atomically

- Bundle ::= [Message | Bundle]*
- OSC_Packet ::= Message | Bundle

- In other words, bundles can hold a sequence, where each element is either a (nested) bundle or a messages
- The top-level packet holds 1 bundle or 1 message

Timestamps

- Every bundle has a timestamp
- Server schedules message delivery
- An example of the Action Buffer or Forward Synchronous paradigm
- Hides network latency
- Need clock synchronization: not fully worked out in current OSC systems (after many years)
- Timestamps are from Network Time Protocol:
  - 64 bit unsigned fixed-point
  - 32 integer bits: seconds since Jan 1, 1900
  - 32 fraction bits (200 picosecond resolution)

Applications

- SuperCollider
  - A software synthesis engine in two parts
    - Server performs audio synthesis
    - Client runs high-level control language
    - Communication by OSC, allows multiple clients
  - Server handles "start", "stop", "compile", etc.
- Open Sound World
  - Another software synthesis system
  - Implements queries so client can discover structure of the server's name space

More Applications

- Interfaces for:
  - Flash
  - Director
  - Perl, Python, SmallTalk
- Various microcomputer sensor systems
- Reactor – commercial synthesizer
- Many installations, networked music systems
- Serpent
- TouchOSC

What's Good About OSC (according to the authors)

- Namespace makes the control points explicit
- Uniform access to all functionality
- Single, extensible access point
- Migrate from single cpu to multiple cpu
- Snapshots of system state automatable
- Polyphonic control through patterns
- Can represent input (controller) data
- Suggests dynamic controller-to-synthesizer mapping

Some Drawbacks

- Client/Server is more restricted than general point-to-point or peer-to-peer system
- String processing/pattern matching overhead (although this does not seem to be a problem in practice)
- Location transparency not fully supported
- Manual entry of IP address, port number
- UDP or TCP: pick one
- Not fully designed and implemented:
  - Query system
  - Clock synchronization
  - Audio streaming (not part of OSC)

RMCP – REMOTE MUSIC CONTROL PROTOCOL

RMCP – Remote Music Control Protocol

- Integrates MIDI and Ethernet
- UDP/IP over LAN
- Supports broadcast-based sharing
- Also has gateway program for WAN
- C and Java, Windows and Linux
- Client/Server Model

Servers and Clients

- Sound Server – messages to synth
- Display Server – animated piano view
- Animation Server – computer graphics
- Recorder – create file from messages
- MIDI Receiver – MIDI in, packets out
- MIDI Station – use computer keyboard and mouse in place of MIDI
- SMF Player – play standard MIDI file
- Player – play file created by Recorder

Connections (or not)

- All servers receive from each client via broadcast messages
- No acknowledgement
- Small programs, reusable

RMCP Network

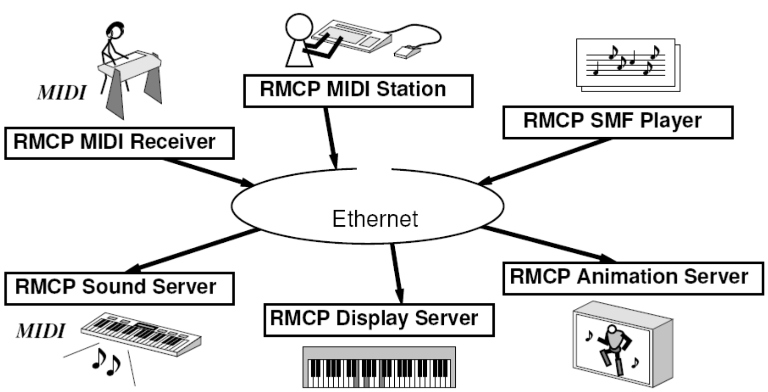


Figure 1: An example of using RMCP servers and clients.

Time

- Early packets held until their timestamps
- Timestamps are optional
- Clock synchronization – requires RMCP time synchronization server
- Every time sync server computes table of time offsets for each machine
- Every time sync server broadcasts table periodically
- Every server listens for local time server's table and uses it to adjust timestamps

Packet Types

- MIDI
- Beat info
- Chord info
- Animation info for transmitting computer graphics

Distribute timing for interactive music systems

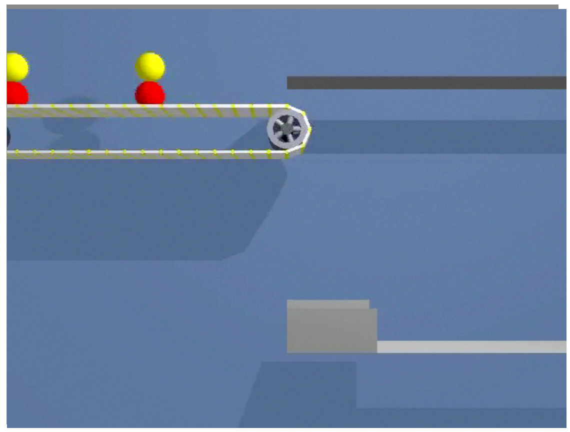
CLOCK SYNCHRONIZATION

Overview

- Why clock synchronization?
- Characterize the problem
- Simple solution
- Some more elaborate approaches
- What next?

Why Clock Synchronization?

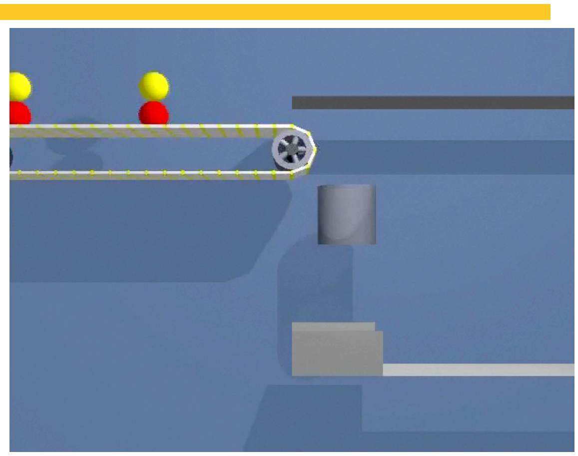
If you have low-latency communication, you do *not* need clock synchronization…



Carnegie Mellon University

Why Clock Synchronization? (2)

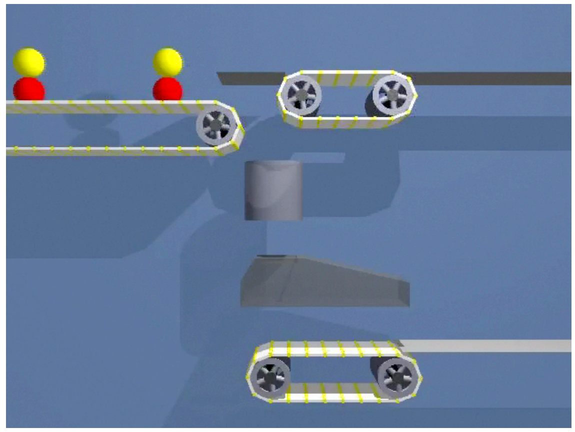
If network communication *sometimes* has high delays (latency), then event synchronization is difficult…



Carnegie Mellon University

Why Clock Synchronization? (3)

Scheduling according to timestamps can overcome some synchronization problems (but not latency problems)…



31

Carnegie Mellon University

Why Clock Synchronization? (4)

- Timestamps are only as good as the local clock…
- …therefore the goal is:
  Synchronize clocks to a precision that is much better than network latency and jitter.

32

Carnegie Mellon University

The Design Space

- What do we synchronize to?
  - Global consensus (internal synchronization)
  - Master reference clock (external synch.)
- Who’s in charge?
  - No one (symmetric)
  - Master (asymmetric, master-controlled)
  - Slave (asymmetric, slave-controlled)
- Special synchronization hardware?
  - Yes: hardware synchronization
  - No: software synchronization

Clock and Network Characteristics

- Crystal clock accuracy: +/-0.02%
- Frequency drift: low
- Network latency: <1ms
- Network jitter: long tail (0.5s)
- Jitter reading clock or frame #: <1ms

- This should be easy…

Network Latency and Jitter

- Interactive music systems
  - not compute bound
  - short or empty network and task queues
  - Messages *usually* get through quickly
- To read remote system time:
  - send message; wait for reply
  - quick reply => low latency and jitter
  - add half of transit time to compensate for latency
  - result should be well below 1ms error

Logical Clock Model

- Assume that time is a linear function of the local clock or sample count:

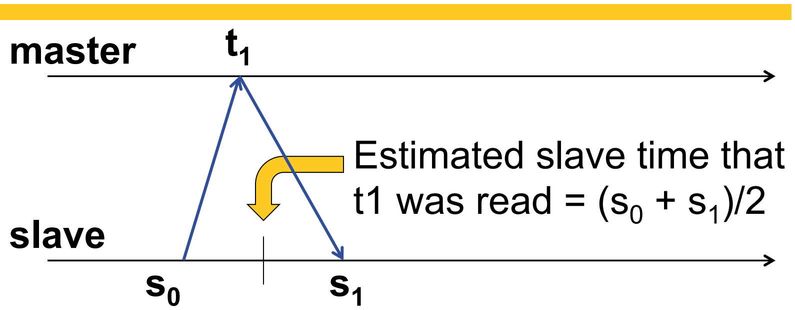
  *LogicalTime = offset + rate * LocalTime*

- Clock synchronization amounts to updating *offset* and *rate.*

Simple Solution

- Periodically read remote “master” clock
- If reply returns quickly, update local time
  - An excellent, robust method:
    - Poll the master clock 10 or so times
    - Find the *minimum* round-trip time
    - Update based on that *single* round trip
- Otherwise, continue with previous model until next period.

Simple Solution



At slave’s local time s, slave estimates master time to be $s + t_1 - (s_0 + s_1)/2$

More Elaborate Approaches

- Dominique Fober:
  - Use window of recent timestamp messages
  - Reject outliers, estimate offset and rate
  - Use exponential smoothing
- Brandt and Dannenberg:
  - Treat logical clock as feedback control system
  - In simulation, achieved 1.1ms clock error with 5ms error reading sample clock.

Clock adjustment

- In O2:
  - If off by >1s, just set the time (local time jumps)
  - Otherwise, run 10% faster or slower until convergence
  - No “right” answer: either you introduce more absolute error or more “jumps” – both are bad.
- How do you deal with unmatched sample rates?
  - Resample?
  - Ignore it and work at control level?

Conclusions

- Clock synchronization is critical for networked interactive systems
  - *Assuming that network latency is significant!*
- Clocks and networks have almost ideal properties.
- Simple approaches work well to ~1ms.
- Advanced techniques can achieve near-frame accuracy over ordinary networks.

Extending Open Sound Control to IP-based Networks

O2

Why O2?

- OSC has been *very* successful – clearly a need for communication support
- OSC designed for flexibility and low-cost:
  - Designers did not want to make assumptions about underlying *transport* mechanism
  - Result is that OSC cannot take advantage of TCP, message broadcast, other IP capabilities
- Computing has advanced:
  - Even phones run IP
  - $10 for a low-powered linux computer!
  - WiFi is everywhere
- OSC shortcomings…

Main Advantages of O2

- Clock synchronization and accurately timed delivery (as an option)
- Choice of reliable delivery vs. lowest latency best effort
- Named “services” instead of IP address and Port number
- Also:
  - O2 has a scheduler that applications can use
  - O2 has OSC compatibility options

O2 Addresses

- In OSC, you form an address, such as

  /voice/3/freq

  and you deliver it by sending a packet to an IP address, e.g. 128.2.42.57 and port number, e.g. 8001
- In O2, you prefix the address with a "service name", e.g.

  /synth/voice/3/freq
- O2 automatically "discovers" the "synth" service

O2 API in Serpent

- `o2_initialize("test", o2_debug_flag)` – join the "test" application (all applications have a name to avoid interference with other O2 applications sharing the network); the debug flag is currently ignored
- `o2_service_new("server")` – create a local service named "server". Messages beginning with /server will be delivered here.
- `o2_method_new("/server/fn", "i", 'sv_fn', t)` – add a handler for "/server/pitch" messages. The messages will contain one integer ("i"), and `sv_fn` will be called to handle the message. t means coerce non-integer parameter, e.g. if sender sends a float, it will be coerced to an int before calling `sv_fn`.
- `o2_clock_set()` – become the clock "master". One host running O2 should do this to establish a global clock.
- `o2_poll()` – must be called frequently to handle O2 protocols and dispatch timed message delivery. Automatic in sched if you set `sched_o2_enabled = t`

Sending Messages

- `o2_send_start()` – begin constructing an O2 message
- `o2_add_int32(i)` – add an integer parameter
- … you can add more parameters here …
- `o2_send_finish(time, "/server/fn", tcp_flag)` – send the message with timestamp (0 means as soon as possible) to address. Tcp_flag is true for reliable (TCP) delivery.
- Address can begin with "!", e.g. "!server/fn", if there are no wildcards in the address – bypasses pattern matching at the server

O2 clock and service setup

- O2 is not immediately fully operational after o2_initialize() – needs clock sync and service discovery.
- Here's a cheap-and-dirty "wait for setup" loop that, as a client, waits until we have clock sync and discover the service named "server":

```
// poll until client is ready to go
while o2_status("server") < O2_REMOTE
    o2_poll() // run o2 while waiting
    time_sleep(0.01)
```


Carnegie Mellon University
ⓒ 2019 by Roger B. Dannenberg

OSC compatibility

- `o2_osc_delegate(service, ip, port, tcp_flag)` - Create an O2 service, named by the string service, that forwards O2 messages to an OSC server defined by the string ip (IP address or "localhost"), the integer port (port number), and the boolean tcp_flag which specifies whether to connect via UDP or TCP. (Now O2 processes can send to “service” to reach the OSC server at ip:port.)
- `o2_osc_port_new(service, port, tcp_flag)` - Create an OSC server that forwards incoming messages to the O2 service named by the string service. The service is offered on the port given by the integer port, and the port will receive messages via UDP unless tcp_flag is non-nil, in which case TCP is used. (Now OSC clients can send to this host’s ip address at the given port to deliver O2 messages to service.)

INTERCONNECTED MUSIC NETWORKS

Taking a Step Back: Why Networks?

- We could talk about esthetics of:
  - Acoustics vs. computer/electronics
  - Computer/algorithmic composition
  - Fixed recordings vs. live performance
- But I picked Networks for several reasons:
  - We're talking about network technology
  - We're aiming for a networked orchestra performance
  - Networks highlight latency and timing issues and concepts which have wide application
  - Networks are one way to enable modular systems, e.g. using TouchOSC, Synthesis servers, etc. – again with wide applications to music, art installations, etc.

Interconnected Music Networks

- A fundamental aesthetic concept in IMNs is the computer's role as a supporter and enhancer of live musical interaction with its surprise, immediacy, and flexibility.

See G. Weinberg, "The Aesthetics, History, and Future Challenges of Interconnected Music Networks"

Cage and Imaginary Landscape No. 4

- “Process” Music
  - A reaction to formal structure in 20th C.
  - A precursor to algorithmic composition
- Cage gives instructions to performers, but sound is indeterminate radio broadcasts
- Chance operations further remove Cage from direct control over sound

League of Automatic Music Composers and The Hub

- Pioneering work in the 1970’s
- Interconnected microcomputers
- Each computer ran a program to generate sound
  - Parameters of the generation process were transmitted to other computers
  - Incoming parameters from other computers affected the generation process

League of Automatic Music Composers and The Hub

55

Carnegie Mellon University

The Bridge Approach

- Network for communication
- Usually video and audio
- Sometimes MIDI
- Used for master classes, rehearsals
- Often latency is a big concern

56

Carnegie Mellon University

The Shaper Approach

- Users manipulate parameters that control music generation
- Music reflects collective input of everyone

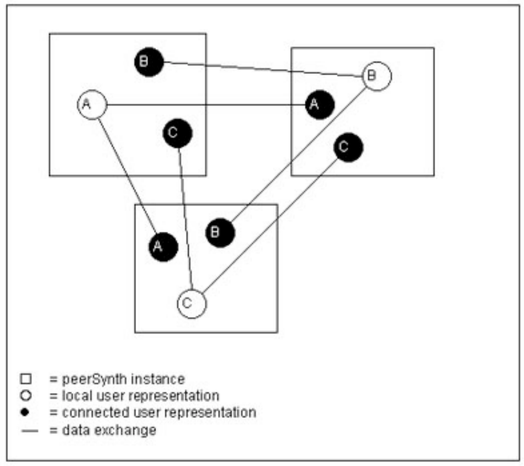
Construction Kit Approach

- Users download musical materials,
- Work on the material, and
- Upload results of manipulation

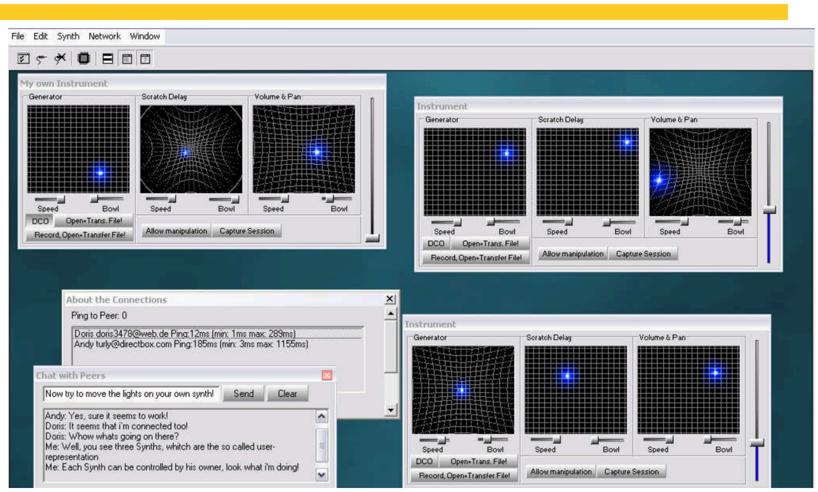
- Sergi Jorda's Faust Music Online is an important example

peerSynth

- Transmit control info only
- Local synthesis
- Treats latency as synthesis modulation parameter



peerSynth



Listening to Examples

- https://www.youtube.com/watch?v=oPfwrFI1FHM (Cage)
- http://www.youtube.com/watch?v=6APygFQ6BAo (Jorda)
- http://www.youtube.com/watch?v=czV9sSGpeyk (LOL)
- http://www.youtube.com/watch?v=eqGo7qRaDZ0 (Oliveros)