Comunicação cliente-servidor de baixa latência com Mobile

Guilherme Freire Silva

Trabalho de Conclusão de Curso
Instituto de Matemática e Estatística
da
Universidade de São Paulo

Orientador: Prof. Dr. Marco Dimas Gubitoso

São Paulo, 2016

Lista de Figuras

Lista de Tabelas

Sumário

1 Motivação

Conhecer o desenvolvimento de um aplicativo para Android, Entender como se dá a comunicação Cliente-Servidor, Fazer uma transmissão de dados eficiente o aplicativo e o servidor.

2 MAPA GERAL DO QUE TA COM TESENO

TODOs: ¿ Explicar o ganho de utilizar JS. ¿ Estrutura Cliente-Servidor ¿ Network Socket ¿ Websocket ¿ Socket.io ¿ Explicar node.js

3 Comunicação Web e Aplicações Web

A comunicação de dados entre computadores através da Internet é algo muito comum, e necessita de uma arquitetura de rede adequada para ocorrer.

(TODO explicar TCP e/ou TCP/IP)

A mais utilizada é o modelo cliente-servidor. Neste modelo, existem dois tipos de processos rodando, o Cliente e o Servidor. O Cliente é o processo que roda nos computadores locais, e se conecta ao Servidor. A comunicação é dada quando o Cliente manda uma requisição ao Servidor (esta requisição pode ser uma página web, um processamento de dados, entre outros). Por sua vez, o servidor recebe esses dados e faz o processamento necessário para completar essa requisição, para em seguida enviar uma resposta ao Cliente, que pode ser o que foi requisitado ou outro tipo de mensagem, como de erro, caso ocorra uma falha no processamento, ou negação de permissão. O Cliente então pode continuar com o seu próprio processo.

(TODO a requisição é um HTTP, explicar HTTP)

(TODO nota de rodapé) A outra grande arquitetura muito utilizada também é a peer-to-peer (P2P). Nessa arquitetura, não há um computador central para funcionar como servidor, cada computador conectado (Peer) na rede realiza funções tanto de cliente como de servidor na aplicação que está sendo executada. Essa aplicação tem suas tarefas organizadas e divididas entre os Peers. Essa arquitetura é conhecida principalmente por ser usada para a transmissão de arquivos grandes, como músicas e vídeos. Nessa transmissão, os Peers que tem o arquivo são conectados aos que não tem, e começam a transferência de pequenos pacotes de dados. Esses pacotes não precisam vir ordenados e o Peer receptor os armazena localmente. Uma vez

que a transferência é completada, ele ordena os pacotes e monta o arquivo final. Uma vantagem dessa arquitetura é que a tranferência não é limitada pela capacidade de banda de Servidor, e Peers podem se conectar e desconectar sem que haja problemas para o receptor, o arquivo são será corrompido por eventuais problemas de conexão. Um ponto negativo é que, sem um Servidor, não há um controle de que tipos de arquivos estão sendo transferidos (o que abre uma porta para pirataria) e não é fácil interromper uma transferência, uma vez que ela pode ser composta de milhares de conexões. Outro ponto é que não é fácil de se conhecer a procedência dos dados recebidos, a segurança não pode ser garantida. Esta arquitetura não serve para as necessidades do projeto, a hipótese de uso foi descartada após o estudo de sua estrutura.

Este modelo é suficiente para páginas Web, pois o cliente pede páginas estáticas e o Servidor as fornece sempre que necessário. Mas essa arquitetura não permite um uso mais dinâmico de websites, pois sua estrutura é muito burocrática. Para fazer uma página mais responsiva, com feedbacks a cada ação do usuário e dados novos, seria necessário que Cliente fizesse uma requisição ao Servidor após o usuário ativar algum gatilho (como preencher um formulário ou levar o cursor a algum ponto específico, por exemplo). Como essa requisição é um HTTP (TODO http), ela possui em seu cabeçalho toda a estrutura de dados para ser processado e enviado de volta, também com um cabeçalho bastante carregado. Ao receber a página atualizada, o Cliente ainda precisa atualizar o que for necessário. Para tal, ele precisa, por fim, atualizar a página. Há dois problemas claros nessa estrutura:

- A necessidade de recarregar a página inteira, mesmo que somente uma pequena parcela dos dados tenha sido alterada. Esse comportamento foi criado em um momento no qual não se tinha a necessidade de alterar pequenas coisas, na página, e que cada HTML vindo do Servidor seria uma página completamente nova. A impossibilidade de atualizar dados individualmente remove qualquer dinamicidade desejada.

- O overhead gerado pelas pela estrutura de dados http. Esse overhead cria uma latência alta e faz com que um envio contante de requisições ao Servidor exija muito recurso.

Um exemplo de comportamento impossível com essas restrições é dar um feedback instantâneo para o usuário na hora que ele está preenchendo um cadastro em algum site, como informar se o email inserido já foi usado, ou se a senha possui os parâmetros mínimos necessários de segurança.

(TODO falar de Flash?) (TODO verificar se) A conexão só dura durante o processo de requisição entre ambos, ela é fechada uma vez que a requisição é satisfeita.

Da discussão gerada à partir desses problemas, surgiu o Ajax (Asynchronous JavaScript e XML). Ajax é uma técnica de desenvolvimento Web

usada para fazer requisições ao Servidor para receber dados no background de forma assíncrona, sem precisar recarregar a página inteira. Com ele, é possível receber dados novos e atualizá-los no código sem alterar o resto da página. Isso com essa técnica passa a ser possível atualizar partes de uma página com base em eventos do usuário.

Para ilustrar esse funcionamento com um exemplo simples, basta usar a ferramenta de busca do Google (TODO link). O usuário começa a digitar uma busca e os resultados já começam a aparecer sem a necessidade de atualizar a página, mesmo antes da busca estar completa (TODO imagem). O input do usuário ativa requisições no background para obter e atualizar os resultados exibidos, sem se preocupar se a busca será alterada no futuro. Assim, o usuário recebe um feedback muito mais dinâmico e menos burocrático do que o convencional (digitar a busca completa e ir para uma página de resultados).

Com o Ajax, nota-se uma mudança de paradigmas na Web. Onde antes só havia a noção de página Web, agora começa a se formar uma noção de Aplicação Web. Antes, a maior interatividade possível era algo como um fluxo de telas com dados dependentes da tela passada, mas agora ações de usuários passam a ter resultados instantâneos. Sites com um fluxo de dados muito grande e dinâmico passam a ser possíveis, como Facebook (TODO link), no qual é possível fazer comentários, interagir com usuários e visualizar tanto conteúdo quando desejado sem precisar atualizar a página.

Essa mudança de paradigmas fez com que muitas Aplicações tipicamente executadas em Desktop fossem desenvolvidas para Web, como Chats ou aplicações que exigissem um fluxo de dados muito grande entre o Cliente e o Servidor. O que ficou claro com o tempo é que Ajax não bastava para muitas dessas Aplicações, em específico as que funcionavam em tempo real. Muitas vezes é o Servidor que precisa se comunicar ao Cliente sobre mudança nos dados. O modelo de funcionamento do Ajax não possui esse tipo de interface, então, para fazer aplicações assim funcionarem corretamente, é necessário muito esforço, lutar muito com a linguagem.

Um exemplo mais claro disso é uma aplicação de Chat. Não é possível ter um padrão de quando o Servidor tem mensagens novas, então é necessário que o Cliente faça requisições a cada período de tempo. Se esse período for curto, o fluxo de dados intenso pode se tornar um problema. Se o período for longo, vai contra o princípio de ser uma comunicação em tempo real, pois as mensagens vão demorar mais a serem entregues. Além disso, ao fazer uma atualização síncrona com o Servidor, o próprio modelo de Ajax tem um comportamento de enviar as mensagens em paralelo, o que não preserva sua ordem, outro fator primordial em um Chat. A necessidade de vários tratamentos para uma aplicação relativamente simples como essa funcionar

deixa claro que há necessidade de mais ferramentas para o desenvolvimento Web.

Até esse momento, não era possível para o Servidor mandar dados espontaneamente, o Cliente precisava fazer uma requisição. Foram criadas técnicas para simular essa comunicação bilateral que o ambiente Web não permite, e elas são: Polling, Long-Polling e Streaming.

4 Polling

Essa técnica já foi descrita acima no exemplo do Chat. Ela consiste em fazer o cliente mandar requisições em intervalos regulares de tempo e receber a resposta logo em seguida. Ela foi a primeira tentativa de se contornar o problema, principalmente por ser a implementação mais simples e direta. Essa solução é boa quando se sabe o tempo de atualização de dados no servidor, pois então é possível sincronizar os tempos de requisição e atualização. Porém esse é somente um dos cenários possíveis, dados em tempo real não costumam ser tão previsíveis, então é inevitável que uma parcela dessas requisições seja desnecessária e muitas conexões são abertas e fechadas sem necessidade, em momentos de baixo fluxo de atualização do servidor.

5 Long-Polling

No Long-Polling, o cliente manda uma requisição, e o servidor a mantém aberta durante um determinado período de tempo. Se uma atualização chegar durante esse período, a resposta é enviada ao cliente com os dados. Se o tempo acabar sem que haja mudanças, o servidor envia uma resposta para encerrar a requisição aberta. Essa técnica apresenta melhoras em relação ao Polling, porém, se existe um fluxo alto de atualizações no servidor, ela não oferece nenhuma melhora substancial em relação a ele. Nesse contexto aliás, o Long-Polling pode ser pior, pois pode ficar instável em um loop contínuo de Polls imediatos.

6 Streaming

Com a técnica de streaming, o cliente manda uma requisição, e o servidor manda e mantém uma resposta aberta, que é atualizada continuamente, sem ter uma data certa para ser fechada (um intervalo de tempo pode ser definido, mas normalmente a conexão é mantida indefinidamente). Essa resposta é atualizada sempre que há novos dados no servidor, mas ele nunca manda um

sinal para completar a resposta e encerrar a conexão. O ponto negativo dessa técnica é que, como o streaming ainda é encapsulado no HTTP, é possível que firewalls e servidores proxy possam escolher armazenar a resposta em um buffer, o que aumenta muito a latência da mensagem.

Por fim, todas essas técnicas envolvem requisições e respostas com um cabeçalho HTTP, que contém muitos dados desnecessários para esse uso, gerando latência. Além disso, uma verdadeira conexão bilateral requer mais do que o fluxo de dados vindo do servidor, é necessário também ter o fluxo originado no cliente. Para fazer uma simulação mais consistente com o desejado, muitas soluções hoje usam duas conexões de fluxo, uma para o cliente e uma para o servidor. A coordenação e manutenção dessas duas conexões gera um overhead ainda maior em termos de recursos, além do claro aumento de complexidade do código.

Por todos esses fatos, ficou claro que era necessário ter uma conexão bilateral de verdade. Para isso, um controle maior da transferência de dados era necessário. Esse controle é feito pelos Sockets, mas eles eram indisponíveis ao desenvolvimento Web até então. Então foi criada o Websocket, uma API para fazer a interface com os Sockets de uma conexão.

7 Websockets

8 Sockets

No modelo cliente-servidor uma conexão é dada por dois pontos finais, um no Cliente e um no Servidor. Essa é a conexão de baixo nível definida pelo protocolo TCP. Cada um desses pontos finais serve para mandar e receber os dados em relação ao outro lado da conexão e eles são nomeados como Soquetes de Rede (Sockets). Cada Socket é definido por um endereço de IP e uma Porta (por exemplo, 192.168.0.1:8000), independente se está no Cliente ou no Servidor, e a conexão TCP é definida de maneira única por seus dois Sockets.

A conexão cliente-servidor, considerando os Sockets, é feita da seguite maneira:

No lado do cliente, Primeiramente ele precisa saber o endereço de IP da máquina onde o servidor roda e a porta que ouve (esses dados definem o socket do servidor). Ele então envia um sinal a esse socket para pedir a conexão. Nesse sinal há um identificador, que contem seu próprio IP e uma porta escolhida na hora, para que o servidor saiba a quem enviar as respostas. É importante ter em mente que o cliente não possui um socket ainda, ele será criado caso a conexão seja bem sucedida. No lado do servidor, se não ocorrer

nenhum erro, a conexão é aceita. Neste momento, o servidor cria um novo socket com o mesmo IP e porta local do original, para conectá-lo com o cliente. Esse novo socket é necessário para que o servidor possa continuar ouvindo a outras conexões naquela porta com o socket original, enquanto a cópia passa a ser exclusiva ao cliente. De volta ao cliente, se a conexão é aceita, o seu socket é criado (com as especificações que foram enviadas no pedido).

Agora o cliente e o servidor podem se comunicar com leitura ou escrita nesses novos sockets criados.

9 Device

Device: O aplicativo é criado usando o Cordova. Cordova é uma IDE (?) que faz o porte de programas em Javascript para várias plataformas, como Android e iOS.

¿ Tiveram mil problemas até o Cordova ser escolhido.

O device se conecta ao servidor utilizando Socket.io

Estruturação de um código Web (JavaScript, CSS, HTML5)

10 Server

Server: Usa socket.io para fazer a comunicação com os clients. ¿ Usa node.js para something

11 Browser

Browser: Primeiro client, lançado pelo server; Recebe dados por Socket; Não envia dados;

Visualização de dados; Visualização 3D: Usa Three.js (Biblioteca 3D para js); Usa os dados recebidos para alteração de objetos 3D.

Foram feitas duas páginas de Three.js: 1) Um simples objeto é rotacionado e tem sua cor alterada de acordo com dados recebidos.

2) É a aplicação de um algoritmo de ruído para geração procedural de terreno em um grid. Utiliza os dados recebidos para alterar os parâmetros do algoritmo. No teste inicial ele altera em tempo real o grid, de forma que o envio constante de dados cria um movimento também constante.

¿ Talvez possa fazer um teste de performance, e rodar o algoritmo em um grid muito maior.

12 Parte Subjetiva

Parte Subjetiva:

Evolução do Projeto:

Esse projeto utiliza várias tecnologias que eu nunca tive contato.

Inicialmente a ideia inicial do TCC era usar um Arduino e um conjunto de sensores para coletar dados do meio e mandar de maneira síncrona para um servidor. O servidor então usaria esses dados para fazer algum controle. Numa primeira discussão com o orientador, a ideia foi rapidamente descartada, pelos seguintes motivos:

Os sensores dele normalmente são imprecisos; A montagem de um dispositivo como o idealizado seria desnecessariamente custosa e muito propensa a erros; Sem uma montagem muito bem executada, o dispositivo ficaria muito frágil; É possível conseguir muitas leituras de sensores utilizando um smartphone.

Com os argumentos apresentados, a decisão foi criar um aplicativo para Android com o mesmo propósito. Todos os argumentos apresentados são resolvidos com essa solução. A implementação é extremamente mais simples; os sensores são muito mais precisos; a estrutura física já está pronta e é um objeto que já está presente em todo o mundo (o produto final atinge grande parte da população); e, por fim, é muito viável a adição ou remoção de funcionalidades, além de muito mais suporte para a plataforma.

Uma vez que a decisão de utilizar um smartphone, foi necessário saber como implementar um aplicativo que tenha acesso aos sensores. A primeira ideia foi utilizar serviço MIT App Inventor.

MIT App Inventor: É um serviço disponibilizado pelo MIT para a criação de aplicativos. Ele é extremamente didático e inclusivo, sua interface não é dada por linhas de código, e sim por blocos lógicos que se encaixam e formam um algoritmo (na prática, é bastante ruim não poder escrever linhas de código livremente). Sua API possui interface para o uso de sensores, além de outras funcionalidades que não foram exploradas neste TCC, como acesso à ferramenta de reconhecimento de voz e ferramentas sociais, como email, mensagem e Twitter. O serviço é bom, porém possui várias restrições, então é difícil de ser usado.

Com certa dificuldade um primeiro aplicativo de teste foi feito, para pegar valores do giroscópio. O próximo passo seria criar um servidor e estabelecer uma conexão entre ambos, porém as opções de conectividade do App Inventor também são muito restritas, então, dada a dificuldade prevista, eu achei melhor deixar essa plataforma de lado no momento e procurar outras alternativas para a criação de um app.

A ferramenta escolhida foi o Kivy.

13 Kivy

É um framework Open Source de Python. Seu objetivo é o desenvolvimento rápido de aplicações multiplataforma que fazem o uso de interfaces inovadoras, como telas com Multitouch e sensores de movimento. Sua API, por exemplo, lida com eventos de mouse, teclado e toques de tela. Ele também possui um foco na criação de apps com NUI (Natural User Interface). NUI é uma metodologia de interface cuja proposta é ser invisível ao usuário, e apresentar uma experiência natural e intuitiva. Seu principal foco é evitar grandes barreiras durante o aprendizado. Através de decisões de design, o usuário tem um entendimento do software sem grandes dificuldades, à medida que a complexidade da interação aumenta.

[TODO] Para exemplificar a facilidade de desenvolvimento de lógica e de uma interface gráfica, abaixo tem o código para se criar um jogo de Pong.

14 Cordova

Cordova: O Apache Cordova é um framework open-source para desenvolvimento mobile. Ele permite o uso de tecnologias Web, como HTML5, CSS3 e JavaScript. Seu desenvolvimento é multiplataforma, então há uma API de alto nível para acessar módulos desejados, como sensores, arquivos e rede. Isso implica em uma interface abstrata para o uso das features, de forma que código desenvolvido será executado em todas as plataformas oferecidas e o desenvolvedor não precisa se preocupar com os detalhes de cada uma na hora da implementação.

A aplicação é implementada como uma página Web em um arquivo 'index.html' que referencia os recursos necessários, como CSS, JavaScript, imagens e arquivos de mídia. A parte lógica é feita em JavaScript, e a renderização em HTML5 e CSS3. O HTML é enviado para a classe "Wrapper" específica da plataforma escolhida, na qual estão definidos os detalhes de implementação. Ela também tem incorporada um browser nativo, o WebView, que executará o programa Web.

Para a comunicação entre o aplicativo e os componentes nativos de cada plataforma, é necessária a instalação de Plugins. Cada Plugin é uma biblioteca adicional que permite ao WebView se comunicar com a plataforma nativa na qual está rodando. Eles provêm acesso a funcionalidades que normalmente não estão disponíveis em aplicações Web. Essas ferramentas são disponibilizadas para o desenvolvedor através de uma expansão da API inicial, que serve de interface e toma para si os detalhes da implementação em cada plataforma, simplificando o uso para o desenvolvedor. Há um conjunto

principal de Plugins, chamado de Core, que é mantido pelo próprio Cordova. Nele estão contidos os que acessam as principais funcionalidades de um dispositivo mobile, como acesso ao acelerômetro, câmera, bateria e geolocalização. Há também Plugins desenvolvidos por terceiros (em geral pela própria comunidade ativa) que trazem relações com outras funcionalidades, que podem ser exclusivas de uma plataforma. Enquanto no Core há apenas umas poucas dezenas, a lista de Plugins criada pela comunidade oferece centenas com as mais diversas funcionalidades, como um para compras dentro do App e um para enviar notificações para dispositivos vestíveis (Smartwatches, por exemplo). A criação de um Plugin é simples e incentivada, no site do Cordova há um tutorial.

O método de desenvolvimento descrito até agora, focado em várias plataformas, é um dentre dois possíveis Workflows disponíveis no Cordova e seu nome é "Cross-platform" (CLI). Ele deve ser usado se o aplicativo desenvolvido pretende abranger a maior variedade de Sistemas Operacionais mobile, com pouca ou nenhuma ênfase em um desenvolvimento em uma plataforma específica. O segundo Workflow é chamado "Platform-centered", e deve ser usado se o projeto é focado para uma única plataforma e se pretende modificá-la em baixo nível.

A arquitetura interna está descrita no diagrama:

15 Bibliografia