VisConnect: Distributed Event Synchronization for Collaborative Visualization

Michail Schwab, David Saffo, Yixuan Zhang, Shash Sinha,
Cristina Nita-Rotaru, James Tompkin, Cody Dunne, and Michelle A. Borkin

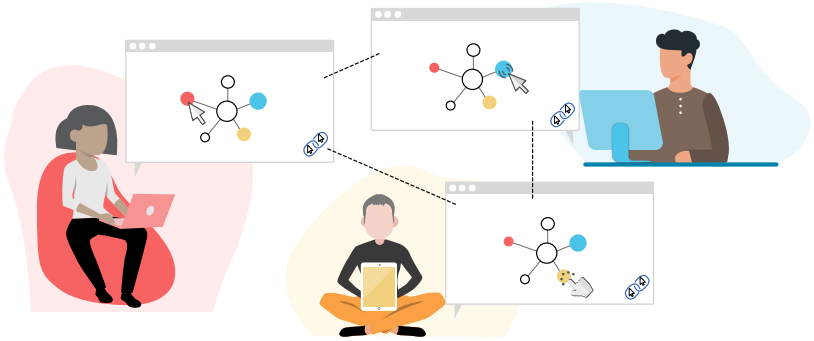


Fig. 1. Three distributed collaborators analyze a node-link network visualization, with each synchronously interacting by dragging nodes. This was enabled by adding two lines of VisConnect code to an existing D3.js node-link visualization. As the world becomes more reliant on collaborating remotely, systems like VisConnect help keep data visualization users connected and effective.

Abstract—Tools and interfaces are increasingly expected to be synchronous and distributed to accommodate remote collaboration. Yet, adoption of these techniques for data visualization is low partly because development is difficult: existing collaboration software systems either do not support simultaneous interaction or require expensive redevelopment of existing visualizations. We contribute VisConnect: a web-based synchronous distributed collaborative visualization system that supports most web-based SVG data visualizations, balances system safety with responsiveness, and supports simultaneous interaction from many collaborators. VisConnect works with existing visualization implementations with little-to-no code changes by synchronizing low-level JavaScript events across clients such that visualization updates proceed transparently across clients. This is accomplished via a peer-to-peer system that establishes consensus among clients on the per-element sequence of events, and uses a lock service to grant access over elements to clients. We contribute collaborative extensions of traditional visualization interaction techniques, such as drag, brush, and lasso, and discuss different strategies for collaborative visualization interactions. To demonstrate the utility of VisConnect, we present novel examples of collaborative visualizations in the healthcare domain, remote collaboration with annotation, and show in an education case study for e-learning with 22 participants that students found the ability to remotely collaborate on class activities helpful and enjoyable for understanding concepts. A free copy of this paper and source code are available on OSF at osf.io/ut7e6 and at visconnect.us.

Index Terms—Collaborative visualization, distributed visualization, toolkit.

1 Introduction

The shared use of visual representations of data by people [21] has wide application across domains, yet the problem of supporting collaborative visualization has been described by Thomas et al. as a "grand challenge for data visualization" [42, Ch.3] and as worthy of a call to action by Isenberg et al.: "We [..] urge for a new generation of visualization tools that are designed with collaboration in mind from their very inception" [21]. This need includes interactive visualization and synchronous remote collaboration, such as for people working from home, people spread across geographic areas, or people practicing social distancing to suppress a global pandemic.

- *Michail Schwab, David Saffo, Yixuan Zhang, Cristina Nita-Rotaru, Cody Dunne, and Michelle A. Borkin are with Northeastern University. E-mail: {schwab.m, saffo.d, zhang.yixua, c.nitarotaru, c.dunne, m.borkin}@northeastern.edu.*
- *Yixuan Zhang is now with the Georgia Institute of Technology. Email: yixuan@gatech.edu.*
- *Shash Sinha and James Tompkin are with Brown University. E-mail: ssinha11@cs.brown.edu, james_tompkin@brown.edu.*

Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxxx/TVCG.201x.xxxxxxx

However, in spite of these calls and this need, there are still few simple-to-use and broadly-applicable systems for distributed synchronous collaborative visualization. While some projects improve information exchange across remote teams [34], most existing literature focuses on co-located collaborations in the same physical space. A key reason why synchronous distributed collaborative solutions are rare is technical complexity: distributed systems are complex, error-prone, and difficult to inspect [31]. Visualization creators often implement customized network communication protocols and distributed coordination algorithms, which end up entwined with the visualization code itself. Such customization reduces generalizability and readability, and leads to 'one-off' solutions. A general-purpose visualization system that simplifies development and provides flexibility and robustness would help lower barriers to the use of remote collaborative visualization.

We contribute VisConnect: a web-based distributed collaborative visualization system. VisConnect replicates browser events, such as *mousemove*, across collaborators to produce synchronized visualizations. VisConnect supports simultaneous interaction with a lock system that gives each collaborator control over specific Document Object Model (DOM) elements. To address the needs of the community, VisConnect simplifies the creation of collaborative visualizations. VisConnect can be used with D3.js [8] and other visualizations based on Scalable Vector Graphics (SVG) on the web simply by including a script tag and applying *minimal* code changes (e.g., replacing `d3.drag` with `vc.drag`). Users then share a URL to the visualization that will connect their browsers in communication. For easy-to-use distributed

visualization development, VisConnect includes drop-in replacements to D3.js functions for collaborative drag, brush, and lasso interactions, and we discuss "divide and conquer" and "all together" strategies to extend other interactions for collaborative use.

We evaluate VisConnect along three dimensions. First, in automated tests, we show that VisConnect can resolve conflicts to ensure that users stay synchronized, such as when collaborators attempt to make different modifications to the same object. Second, we demonstrate the utility of VisConnect with an example implementation for live chat and by presenting three use cases in the domains of health care, remote collaboration, and education. As part of the use case in education, we also evaluate the usability of VisConnect for e-learning through a case study with 22 students on a team-based in-class activity to teach students about network visualizations. Students overwhelmingly stated that the collaboration was helpful and enjoyed the highly interactive remote teamwork enabled by VisConnect. Finally, we identify research opportunities and summarize our lessons learned in how to challenge the single-user assumption often made in visualization data and view manipulation interactions today. A free copy of this paper and source code are available on OSF at osf.io/ut7e6 and at visconnect.us.

2 Related Work

2.1 Distributed Collaborative Systems

Distributed collaborative systems are becoming more commonplace. One might consider video conferencing tools with screen-sharing capabilities as a collaboration tool, but these are asynchronous because they let only one person interact at a time. Collaborative text editors such as Google Docs [16] allow simultaneous edits. For the manipulation of text, an often-used approach is "Distributed Operational Transforms" [13, 15, 39, 40], where simultaneous manipulation operations on the data are merged in a way that a consistent result is generated. This is in contrast to the more conventional approach of resolving conflicts using locks, where parts of a document are "owned" by one collaborator at a time that can make changes. Ellis & Gibbs argue that in the text editing context, the granularity question is hard to answer, as it is not clear whether the lock should be obtained for the enclosing paragraph, just the word, or the character [15]. In typical web-based data visualizations, the DOM structure lends itself well to the use of a lock system, as individual elements can be "owned" by different clients for interaction. In our work, we use a lock service to clearly establish allowed and disallowed operations, and achieve near-zero latency in common interactions by requesting the lock in anticipation of manipulations.

2.2 Collaborative Visualization

Research on collaboration in interactive systems has been ongoing for decades and has been approached through different concepts, including the Abstraction-Link-View paradigm [18]. For visualization, Isenberg et al. define collaborative visualization as "the shared use of computer-supported, (interactive,) visual representations of data by more than one person with the common goal of contribution to joint information processing activities" [21]. The authors categorize the design space of collaborative visualization into *distributed* or *co-located*, and *synchronous* or *asynchronous* according to where and when the usage scenarios happen, respectively. Among these areas, co-located synchronous collaborative visualizations has been most thoroughly studied [4, 12, 19, 20, 22, 28, 30, 44, 45][1], typically using multi-user table tops or multiple co-located devices under the term "beyond the desktop". As part of this work, interesting novel interaction techniques have been proposed, such as collaborative brushing and linking [22]. This collective research demonstrates that collaborative visualizations enable new data visualization research.

In contrast, distributed synchronous collaborative visualization has received comparatively little attention [2, 6, 10]. One related research area considers how to propagate visualization interactions across multiple coordinated views, using constraint-based layouts, data-centric approaches, and methods using the Model-View-Controller framework, where all views share a model that propagates changes [32, 35, 37]. These achieve successful coordination among multiple views, and can change the internal application logic and flow. We cannot rely on this ability if we aim for applicability to many independently-created data visualizations. Another better-established area of distributed collaborative visualization research centers around the use of virtual and mixed reality [7, 14, 23, 29, 47]. Key differences exist between these setups and the web, as they often position users in a shared "world" established by a central server, whereas interactive websites operate more independently. Unfortunately, little work has focused on remote collaboration with 2D display and interaction devices. This is true even though remote work has become increasingly popular and important challenges to remote collaboration have been identified, both for the technical aspect of distributed synchronous work [21] as well as for the coordination of team members trying to achieve a common goal [7].

One approach to ensure that all clients see the same content is to synchronize the DOM across clients. This approach is used by Webstrates [24]: a general framework for synchronizing documents on the web. This method has the drawback of breaking data binding if no specific solutions are provided (Sec. 3.2), which makes this approach impractical as a general-purpose system that can extend existing data visualizations. Vistrates [5] helps address this challenge for data visualization by providing a framework with pre-existing visualizations that can be used in different contexts. Within the Vistrates framework, DOM synchronization is achieved without breaking data binding with specific visualization implementations that also synchronize application state. This approach does not transfer to visualizations created outside the Vistrates framework. Two works are most related to our approach for distributed synchronous collaborative visualization: Thum & Schwind propose co-browsing by sharing browser events [43], and Badam & Elmqvist's PolyChrome uses a similar shared browser event system for distributed visualizations that achieved synchronization without breaking data binding [3]. Their approach uses timestamps to resolve orderings, while we use a consensus-driven approach with a lock system to avoid conflicts. Further, neither work supports simultaneous interaction. Because Thum's synchronization works independently from the visualization, the implemented interactions can not be aware of simultaneous events. PolyChrome is designed to be extended with operational transforms to resolve conflicts from simultaneous interactions, but this was outside the scope of their work. In contrast, our system adds collaboration with simultaneous interactions to visualizations by simply adding a script tag, which lowers the barrier to entry.

3 System Design

Increasing adoption of synchronous distributed collaborative visualizations requires understanding the difficulties faced by visualization creators. We begin by stating these as requirements, before analyzing how well different possible architectures may meet these requirements for the data visualization domain (Sec. 3.3).

3.1 Requirements

To lower barriers for development and use, we seek a design that is applicable to all types of visualization applications and is compatible with existing visualization paradigms. We claim the following architecture requirements:

GENERAL: We seek an architecture that is as generally applicable and works on as many visualization applications as possible.

EASY TO USE: We seek an architecture that lowers barriers and makes adoption easy. Adding collaboration should be as automated as possible, and should ideally only require including a script tag.

COMPATIBLE: We seek an architecture that is compatible with existing methods and paradigms. This lets visualization creators add the system to existing infrastructure without requiring new development paradigms.

Further, as any other distributed system, our system must be *safe* — to apply only agreed-upon changes — and must be *live* — to apply

[1] An overview of research in both co-located and distributed settings is in Section 8 ("Collaborative Immersive Analytics") of "Immersive Analytics" [7].

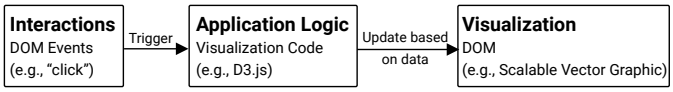


Fig. 2. There are three possible entry points for synchronization within the update cycle of web-based data visualizations. *Interactions*, such as click or drag events; the *application logic* which updates the internal state through JavaScript code based on new information; and the *visualization*, which renders the output. To achieve synchronous collaboration, systems must reproduce this update cycle for all clients.

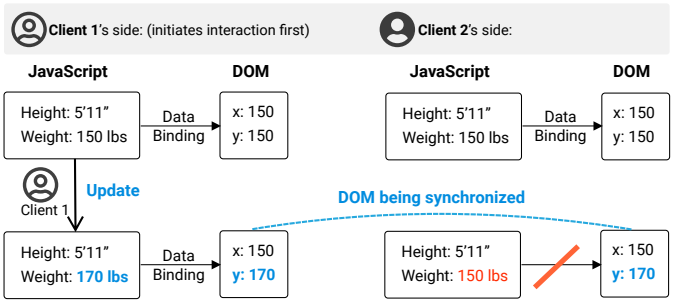


Fig. 3. An illustration of one of the challenges of synchronizing the DOM across clients. Synchronizing the visualization directly conflicts with data binding. Client 2's DOM is updated based on the changes by Client 1, but the data connected to the DOM is now out of date.

changes immediately to be responsive [26]. Balancing both safety and liveness is a core problem in distributed computing [1], and system designers need to make choices about which is more important. For visualizations, we balance these as follows:

SAFE: Given that we seek an architecture applicable to a broad range of applications, we have to assume that in some applications, the display of incorrect information, even if temporary, can prove detrimental. In addition, it is difficult to resolve conflicts without knowing the specific use case. We prioritize "safety" and request access before changes are made to support the general utility of the system and to ease conflict resolution.

RESPONSIVE: Low responsiveness is quickly apparent in many interactions, such as dragging data points [17]. Given that we prioritize safety, we cannot achieve full liveness. However, we aim to mitigate many of the challenges of not being "live" by employing strategies to improve the perception of responsiveness.

3.2 Visualization Update Cycle

To understand how to meet these requirements, we analyze the typical update cycle of data visualizations (Fig. 2). Visualizations are created and maintained by application logic code, often written using JavaScript (e.g., D3.js). This happens through data binding: the application loads data and assigns it to visual objects that represent the data. Any user interaction with the data visualization, such as clicking or dragging a node, is detected by event listeners that are part of the application logic. This causes updates in the data, which the application logic uses to call for the rendering of the visualization to reflect the changes in the data, e.g., via SVG. Each layer in Fig. 2 is a *potential entry point* for collaborative systems to synchronize the visualization across clients. As such, next we discuss the trade-offs of these possible architectures.

3.3 Architecture Choices

Visualization Synchronization An obvious choice is to synchronize the visualization itself. Since most web visualizations are based on SVGs, this represents a well-defined and relatively small target specification to be replicated across clients. This means that comparably little implementation work would be needed to adapt visualizations to a collaborative setting. In this case, a system could guarantee consensus on a shared DOM among clients and keep the local elements up to

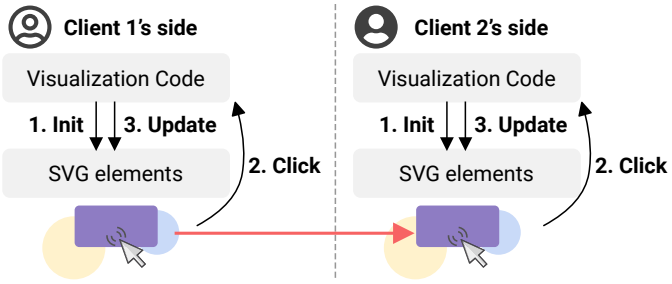


Fig. 4. Event synchronization can achieve the same visual update across clients: From an initial starting visualization across clients (1), a click (2) by client 1 makes the corresponding visualization code respond and update the visualization accordingly (3). If the click event (2) is broadcast to client 2 and then re-emitted, client 2's visualization code will react as per client 1 and update the visualizations synchronously.

date, thus ensuring that collaborators see the same visualization. This approach is used by Webstrates [24] to synchronize document contents.

The disadvantage is that it conflicts with the concept of data binding (Fig. 3, COMPATIBLE). The conflict arises because this approach sets DOM attributes directly, such as element positions, whereas data binding changes attributes to reflect the data. For example, the position (x, y) of a circle on a scatter plot could represent the values for the height and weight of a person. With data binding, x and y are updated when the height or weight change. With a visualization synchronization approach, x and y are set directly without a change in the data, and so the client's application logic would not know about the data change. This is a problem because application logic typically assumes sole responsibility for managing the DOM. If the visualization code re-renders the visualization, any changes made by such a synchronization system would be overridden by the visualization because the data would still be out of date. Collaboration systems using this approach would have to find a solution to the problem of shared responsibility over the DOM between the application logic and the system.

Application Logic Synchronization Achieving synchronization of application logic across clients would not break data binding, as the same executed application logic would lead to the same updated data and the same updated DOM. This could be achieved via custom events, such as with `d3.dispatch`. For example, based on user interaction, a visualization may trigger a "data-updated" event that declares the updated data. This event can be distributed to connected clients so that all update their data accordingly, and developers can control what is communicated to different clients.

The disadvantage of this approach is that it is visualization-specific, and a lot of work has to be done on an individual visualization level to enable collaboration. Specifically, visualizations have to be structured around these custom events so that changes from other clients always trigger the correct update procedure on all clients. Any updates that are executed in response to local changes, such as `onclick` listeners, can lead to inconsistent visualizations. Therefore, this approach can easily lead to errors if not implemented correctly. Collaboration systems using this approach are potentially useful and can have customized behaviors, but require more experienced developers that are willing to invest time in customizing their visualization code for collaboration. Little of this work is generalizable to other visualizations, which does not lower barriers and increase adoption (GENERAL, EASY TO USE).

Interaction Synchronization Synchronizing the execution of low level DOM events, such as `onclick`, `mousedown`, `mousemove`, and `mouseup` causes the entire data visualization update cycle to be executed for all clients (Fig. 4). This is advantageous because this approach also has a well-defined and small target specification (which minimizes visualization implementation changes), and because it is compatible with data binding. One way to think of this is that it allows multiple simultaneous cursors (and other interaction types). As most web data visualizations already react to DOM events, they do not have to be changed to exploit this approach.

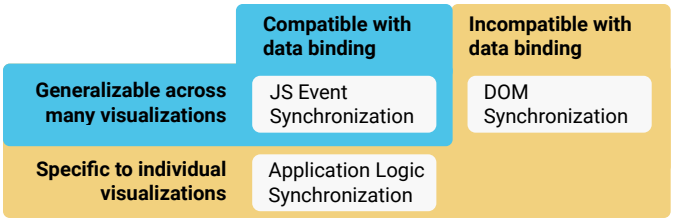


Fig. 5. Possible data visualization collaboration system architectures. *Bottom left:* Application logic synchronization gives flexibility to visualization developers, but ties collaboration code to specific visualization code which hurts reuse. *Top right:* Synchronizing SVG DOM visualizations is more generalizable, but breaks the data visualization update cycle as it is ignorant of data binding. *Top left:* Synchronizing user interactions is general because DOM events are standard browser events, and it is compatible with data binding because any updates to the visualization flow through the application logic. On the other hand, depending on implementation, the flow through the application logic may not always happen in exactly the same way or have the same results. This is a challenge in the JS Event Synchronization approach.

The disadvantages of this approach lie in its assumption that collaborators have equal starting points and that they have equal outcomes from the same interactions. This assumption can be violated for a variety of reasons, including randomness of the visualization, or different screen or window sizes. Some visualizations assume that only one cursor exists, and may not react correctly when used with multiple cursors.

Yet, these challenges can be addressed, and are outweighed by three core advantages: little implementation work is needed to adapt visualizations for collaboration, this approach scales to many visualizations, and the approach is compatible with data binding.

Summary For a system to be easy to develop with, compatible with data binding, and to synchronize entire data visualizations (EASY TO USE, COMPATIBLE), we argue that synchronization of events is the most suitable approach (Fig. 5). This approach is especially useful to generalize to many visualizations (GENERAL), both existing and new, and to have the largest potential community impact. Of course, collaboration systems with more specific requirements may choose application logic synchronization. However, for the rest of this paper, we will consider the technical challenges that come with interaction event synchronization, and propose our solutions through VisConnect.

4 VisConnect System

VisConnect is a peer-to-peer distributed system which synchronizes low-level pointer events, such as `onclick` and `mousemove` (Fig. 4, COMPATIBLE). This is challenging to achieve in a way that leads to consensus among clients, as different collaborators can try to execute conflicting modifications to the data and the view at the same time. We address this with explicit client leadership, DOM partitioning, and an interaction lock service.

Practically, we implemented VisConnect for web-based SVG visualizations. The implementation works across mobile and desktop computers, is open source[2], and is written in TypeScript using Peer.js. This approach allows it to be easily be added to many existing visualizations with only a few lines of code (EASY TO USE). To help in this endeavor, VisConnect also includes a customizable set of JavaScript events that are synchronized across clients.

4.1 Overview

The main functionality of VisConnect is contained in the Core module, shown in Fig. 6. This intercepts DOM events with the DOM module, and asks the Synchronization module how to handle them. The Synchronization module decides which events to allow and which to reject based on the lock service. Then, it broadcasts events to other clients using the communication component, which also sets up the network,

[2]Source code is available at github.com/michaschwab/VisConnect

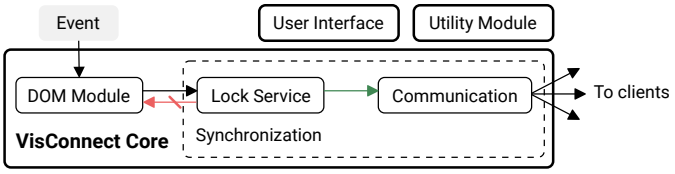


Fig. 6. System design of VisConnect. This consists of a minimal user interface, utility functions for visualization developers, and the Core that distributes events among clients. Events are picked up by event listeners in the DOM module. These are then sent to the Synchronization module, which decides whether to allow or reject them based on the response of the lock service. Allowed events are broadcast using the communication component, which is also responsible for managing the clients.

catches up newly joined clients, and transmits lock ownership information to the clients. If the Synchronization module decides that an event should be emitted, whether local or remote, the Core module uses the DOM module to re-create the event based on the transmitted event information, and dispatches the event on the original target element as identified via element selectors.

The components are written in a modular way to allow for individual components to be swapped out for alternative implementations. The provided user interface and utility module are not needed to run VisConnect — visualizations can create their own interface for their collaborative visualizations (GENERAL). The Synchronization module is not linked to the DOM module, but rather decides which events to execute and which ones to reject, without any knowledge of the implications. As such, the Synchronization module can be tested automatically in isolation with arbitrary mock events to verify system safety (SAFE). The communication component can be swapped out for a mock communication component that avoids the tests to require a network, allowing consistent tests independent of network conditions. The source code includes automated tests of conflicting changes that show VisConnect's correct handling of these situations.

4.2 Interaction Synchronization

The synchronization component is responsible for synchronizing events across clients. This component receives events from both the network component (events from other clients) and the event listener component (events from the local client). Events received from the event listener are processed, checked for access with the lock service, and if approved, broadcast to all other clients. Events received from other clients are processed and executed or ignored, depending on whether the sending client has been granted the access to execute events on the passed element, as determined by the lock service. Events from both sources are recorded in a event log.

Coordination VisConnect uses one leader per collaboration session. The leader's responsibilities are to (1) manage the events, (2) manage the clients, and (3) manage the locks, which we defer to Section 4.3.

(1) Managing the events: The leader keeps track of the events that have been accepted to occur on the network, and keeps track of the order of the events per element.

(2) Managing the clients: All collaborators can invite new clients to join the network. Upon joining, a new client requests to join the network by sending a message to the leader. The leader responds by sending the list of connected existing clients, and the new client establishes connections to them.

In the case that a client joins the network after events have already occurred, the leader ensures that all clients are in the same state by sending the list of events that have occurred up until that point. The new client then executes them in the correct order. This catch-up process is primarily limited by message size and event timing. When all `mousemove` events are captured for several minutes of continuous interaction, the array can become too large for PeerJS' serialization of the data into JavaScript Blob objects. Often, the entire list of `mousemove` events is not necessary to reproduce a visualization state, and better summaries of the past events could help address this limitation, such

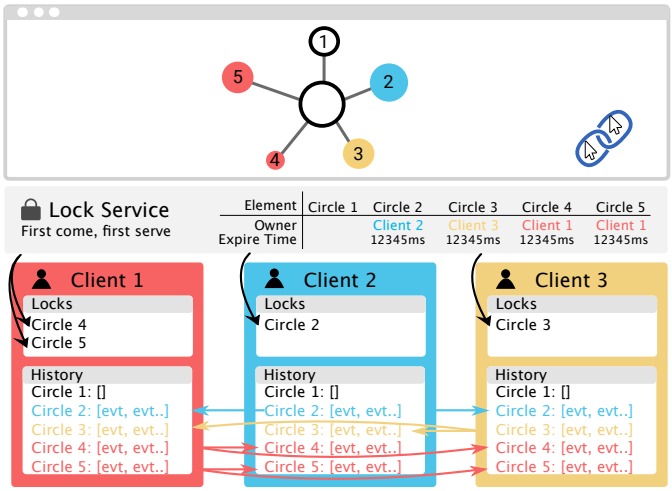


Fig. 7. The internal states of connected clients and the lock service during a collaboration (Fig. 1). On a first come, first come basis, the lock service grants collaborators write access to individual elements, which can then trigger events across all clients. After a short time of not broadcasting events, the write access is revoked so other collaborators can start modifying the element. This design ensures that no conflicts occur between collaborators, while allowing smooth simultaneous interaction.

as custom events. For some interactions, the event timing matters for the visualization state, and it is not possible to reproduce the same visualization state by replaying the events faster. For these cases, a better summary of the past events would be helpful, too. VisConnect does not yet implement custom summaries; this is left for future work.

Partitioning To establish a series of events to be executed on the visualization across all collaborators, some logic is needed to define which events are allowed to be executed and which are prohibited from execution. For example, if two collaborators try to drag the same DOM element, a system design needs to define how to proceed and how to establish consensus among these collaborators. One common distributed systems method to support synchronous changes while avoiding conflicts is to partition the data on which these events operate. This technique is used in most collaborative systems [33], including Google Docs, where the text is separated into underlying sections and paragraphs of text. VisConnect translates the concept of partitioning to the data visualization domain by partitioning the visualization via its individual SVG elements. This allows one element to be manipulated by one client at a time, avoiding ambiguity on this element, while enabling simultaneous changes on other elements by other collaborators.

4.3 Lock Service

While DOM elements are partitioned, the VisConnect Synchronization module still must disambiguate which client can manipulate which SVG element to avoid conflicts (Fig. 7). To fulfill the safety requirement (SAFE), the lock must be obtained before any events can be dispatched by the client. If the element is already controlled by a different client, then the lock service must reject the request, which leads to other clients not accepting events from that client for that element, even if broadcast. If the element is not yet controlled by a different client, then the lock service must accept the request and broadcasts to all clients that this client is the lock owner of the element. Then, other clients will accept events from that client on this element.

A common method of achieving consensus is with a vote-based lock service, via a voting system similar to PAXOS [27]. In this setting, a majority of the clients must agree to granting the lock to a client. However, we found this approach to be too slow for our interactive visualization setting, as the system would have to wait for communication from at least half the collaborators. The delay compounds as the number of clients increases.

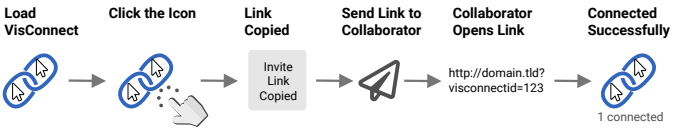


Fig. 8. How to use the VisConnect user interface. A click on the logo copies the link for inviting others to join the collaboration. After the copied link is shared with a collaborator, the collaborator can join the session by clicking the link. Then, the user interface indicates the number of collaborators with "1 connected". The VisConnect user interface is shown in the bottom right of the browser window on a VisConnect visualization.

Instead, we use a central lock service that is conceptually separated from the rest of the system. The host collaborator automatically also serves as the lock service. Clients request the lock from this central source, which minimizes delay and the number of required messages for the decision to be made. This reduced communication time and so additionally led to reduced conflicts in our interactive setting.

For smooth interactions (RESPONSIVE), we request the lock as early as possible. Specifically, any events on an element will request the lock, including mouseover events. For example, in a drag interaction, the lock will be requested even before the mouse is pressed down, making the delay unnoticeable. Once the lock is obtained, the agreement is that the client is the lock owner for at least one second. This means that the client can execute and broadcast subsequent events without communication or delay. In addition, each broadcast event extends the lock ownership for one second. This means that once a lock is obtained, the client can indefinitely smoothly interact with the element without requiring approval. Once the collaborator stops interacting with the element, the lock expires after one second, freeing the element to be interacted with by other collaborators.

This one-second lock avoids conflicts among collaborators, and serves as a "grace period" for lock owners in cases of accidental control misrelease, e.g., during mouse repositioning. We determined the lock's duration empirically: we try not to release an element too early or too late; longer durations could lead to cumbersome element owner changes. This duration will be a parameter in a future VisConnect.

In some visualizations, conflicts are categorically impossible, such as when all collaborators can independently annotate and draw over a visualization, as in Sec. 6.2, or in our chat example in Sec. 5.2, and there is no sense of elements that could be owned. For these cases, the lock service can be disabled entirely in favor of more responsiveness. As seen in Listing 2, the collaboration can be set to "live" as opposed to "safe" to turn off the lock service.

4.4 User Interface

We provide a minimal user interface (UI) to invite friends and co-workers to collaborate, and to indicate how many collaborators are connected (Fig. 8). Both the host and the collaborators can click the user interface to copy the link to invite other collaborators to join. By default, the UI includes the collaborators' cursor positions to help coordination. We plan to make this a changeable setting in future work and provide more means of non-verbal communication to aid collaborative tasks. The VisConnect Core works independently of this interface, and visualization creators can choose to use VisConnect without using this interface. For example, visualization creators can integrate the button to start collaborations into existing menus for appearance consistency.

4.5 Communication

The communication component connects clients and sends messages between them. We use a peer to peer network implemented with Peer.js [11] and the WebRTC protocol. Aside from a lightweight in-browser Javascript server that Peer.js uses to initially connect clients, the distributed nature of the system design can help provide resilience to network infrastructure issues. It also allows our implementation to be easily added to many visualizations with minimal effort (EASE OF USE): typically, including a script tag suffices to enable collaboration.

The process starts when a user visits a hosted visualization where the network code will check the URL for a peer ID parameter. If there is no peer ID present, the client becomes the host and a URL containing the client's peer ID is created. The host can share this link with other users to allow them to join the network. When a user opens this link, the system detects the peer ID parameter and connects to the host. When the host receives a new connection, it adds the peer to its list of connected peers and sends a 'new connection' message back to the newly connected client. The message contains the peer IDs of all connected peers, and allows the new client to connect to all the other clients in the network. Once a client is connected to all the peers, it can send and receive messages. In the current version of VisConnect, the host is the only point of entry for new clients to join the network, and so collaboration ends if the host crashes. As this happens rarely, PAXOS [27] could be used to choose a new host in a future update.

VisConnect sends as many update messages per collaborator as possible given network constraints. If the network is too slow, VisConnect starts to batch multiple events into a single message to avoid clogging the network. However, given modern internet connections, this does not happen frequently and the network delay is rarely noticeable, even with a relatively high amount of data transfer such as in the e-learning example in Sec. 6.3, where four students per group simultaneously and continuously dragged different elements.

4.6 DOM Module

Visualizations across clients are synchronized by executing events from one client on all other clients. To this end, capturing events on the client is core to any VisConnect implementation. The event listener component is built to listen for a defined list of events on the specified target — typically, the SVG element containing the visualization, or the HTML body. We listen for a variety of interaction events such as mouse move, mouse down, mouse up, and click. This list can be easily expanded to support other predefined or custom events. When an event object is captured it is simplified to exclude any non-serializable data and sent to the VisConnect Synchronization component to be potentially processed, broadcast, and executed.

4.7 Utility Module

VisConnect gives visualizations access to an agreed-upon list of DOM events from collaborators. The availability of events from multiple collaborators has implications for visualization design and interaction, as visualizations are typically designed with only a single user in mind.

For interaction, many common techniques are implemented for a single cursor. For example, `d3.drag` listens to *any* `mousemove` events to move an element once `mousedown` has occurred. If two clients are moving their cursors, and one attempts to drag an element, both clients' cursor positions are picked up, and the element will flicker between the two cursor positions. To support multiple collaborators, traditional techniques have to be generalized. The utility module comes with generalizations to all interaction techniques included with D3.js and more, including `vc.mouse`, `vc.drag`, `vc.brush`, and `vc.lasso`. These interaction techniques are discussed in Sec. 5.1.

Another single-user assumption often made in data visualizations is that if no specific node position or color is needed, such as in a node-link network visualization, then these values can be set randomly. If multiple users try to collaborate on such a randomized visualization, they will have inconsistent starting points for their collaboration, which can lead to conflicts. To address this issue, the utility module comes with `vc.random` as alternative to JavaScript's `Math.random`. This function provides random values for the leader of a collaboration, but yields the same values for collaborators that join the session. This ensures that collaborators have the same starting point and continue to see the same visualization during a collaboration. D3.js includes some utility functions, such as `d3.randomNormal`, for creating random values that follow specific distributions. By default, these are based on `Math.random`. However, visualization creators can change this source: Setting `d3.randomNormal.source(vc.random)` makes D3.js' random functions compatible with VisConnect collaboration.

```
const svg = d3.select("#vis");
const circles = d3.range(20).map(() => (
  {x: vc.random() * 500,  y: vc.random() * 400}));

const drag = vc.drag()
  .on("drag", function(d) {
    d3.select(this).attr("cx", d.x = d3.event.x)
      .attr("cy", d.y = d3.event.y);
  });

svg.selectAll("circle")
  .data(circles).join("circle").attr("r", 20)
  .attr("cx", d => d.x).attr("cy", d => d.y)
  .attr("fill", (d, i) => d3.schemeCategory10[i%10])
  .call(drag);
```

Listing 1: Collaborative, draggable circles. To add collaboration to the existing visualization, `Math.random` and `d3.drag` are replaced with `vc.random` and `vc.drag`, respectively. Live demo at bl.ocks.org/michaschwab/a05187daaaa652de0e903365d1f0d327.

In most cases, no additional changes beyond changing `Math.*` and `d3.*` to `vc.*` are required to switch to VisConnect's extensions that support collaboration. But in some cases, the visualization implementations also include single-user assumptions. For example, code might include global variables such as `dragCircle` that are modified when dragging occurs. In these cases, implementations must remove their single-user assumptions to use collaborative drag.

5 VisConnect Use and Interaction

To use VisConnect, a developer must add a script tag to the HTML page of a visualization. This must come before the JavaScript code that creates the visualization, so that VisConnect can capture events before they reach the visualization. After adding the library, collaboration can begin. A click on the VisConnect UI, shown in Fig. 8, lets users share a link with their collaborators. Once they click the link, they join the collaboration session and events are synchronized among collaborators.

5.1 Interactions

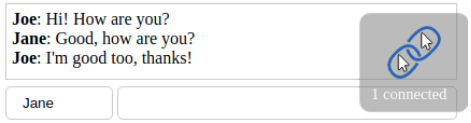
VisConnect supports all interactions in the D3.js API, with many shown on our website at visconnect.us and in our Supplemental Video. By default, mouse events are copied across collaborators. This works well for collaboration scenarios where all collaborators are meant to share the same visualization state.

Selection, Zoom, and View Navigation Consider a map visualization that might zoom in to a clicked country. If collaborators wish to all look at the same country at the same time, perhaps to verify or discuss each other's findings, then the strategy of copying mouse events is suitable. Similarly, this strategy works well for manipulating the visualization state through a user interface, where either the view or the data itself are manipulated through sliders, drop-downs, input fields, and more. We show an example of shared state through shared details-on-demand hover tooltips and shared data alignment settings in our IDMVis example in Sec. 6.1 and our Supplemental Video.

Drag To support drag, collaborators must *not* share the entire visualization state: if "dragging data point 5" is common across collaborators, then they would compete to drag the same data point. Thus, we replace `d3.drag` with `vc.drag` to enable multi-user drag. Listing 1 shows a compact example of collaboratively draggable circles with minimal changes from an existing demonstration of D3.js drag.

Brush and Lasso These selections are similar to drag interactions in that sharing the entire visualization state would break the interactions because the selections would be in conflict between multiple cursor

```
<body collaboration="live" custom-events="chat-msg"
 ↪  ignore-events="click,mousedown,mouseup,mousemove">
    <!-- Messages and input fields -->
</body>
const onMessageSent = () => {
    let evt = CustomEvent('chat-msg', {sender,text});
    document.body.dispatchEvent(evt);
};
const messages = [];
document.body.addEventListener('chat-msg', (e)=>{
    messages.push(e.detail); rerender();
});
```



Listing 2: A simple chat created with VisConnect based on custom events. Mouse events are ignored, but custom chat message events are synchronized across clients. Live demo at bl.ocks.org/michaschwab/20d5f1e5daa091e699f5de4be662ea7d.

positions. Collaborative brush and lasso are more complex than collaborative drag: the simultaneous drag of multiple items has an obvious resulting shared state in the new positions of the data points, whereas the resulting state for multiple brush or lasso selections is less clear.

One important question is whether the result of multiple brush or lasso selections should be a shared state common to all collaborators, such as a shared overall selection of items, or whether each collaborator should have their own state, such as an individual selection. For a shared overall state, we must decide how to combine different selections. For example, one use case could be to select all items within *any* lasso. With `vc.lasso`, we include this as a selection strategy and demonstrate it online[3] as well as in the Supplemental Video.

On the other hand, a collaborative brush and lasso could "divide and conquer", where collaborators individually analyze separate parts of the data to contribute to a shared overall goal. Isenberg et al. [22] define this as "an awareness technique, in which the interactions of one collaborator on a visualization are visible to other collaborators viewing the data items in their own visualizations or views of the data." We support this strategy in `vc.brush` and `vc.lasso`, as shown in the lasso bl.ock above as well as in our collaborative brush demo[4]. With this strategy, visualization creators must decide how to visualize individual collaborators' work as progress on the overall task. We provide an example, "Where's the Square"[5], where collaborators can individually zoom in to different parts of a large scatter plot and annotate for all collaborators to see whether the selected part of the data includes a target data point. This strategy transfers to more complex tasks.

Currently, VisConnect supports most interactions commonly used in data visualizations, with active work to support a broader range of scenarios. We discuss these and other strategies to extend single-user interactions to collaborative use cases in the Discussion (Sec. 7).

5.2 Custom Events: Chat Example

To customize collaboration behavior, VisConnect can synchronize user-defined custom events. As discussed in Sec. 3.3, custom events can be used in JavaScript for a more loose coupling between the input events, such as `mousemove` and `click`, and the effect, such as creating data points and updating the visualization. The loose coupling has the advantage that the input mechanics can be changed without having to change the way the visualization reacts to new data. For example, a set of user interactions, such as clicking, dragging, and entering values, can trigger a "data-created" event, which can be captured to update a visualization. Custom events are supported by D3.js using `d3.dispatch`. An example of custom event synchronization is shown in Listing 2: a chat based on VisConnect. When collaborators submit chat messages, custom "chat-msg" events are broadcast. The list of messages are updated when new chat-msg events are received. By synchronizing these custom events across clients, the communication of chat messages happens automatically through VisConnect.

In the custom-events HTML attribute, users can specify which events should be shared among collaborators. In this example, the cursors should not be synchronized, and so we specify not to synchronize the corresponding events with the ignore-events tag. Allowing users to specify which custom and standard events are synchronized allows visualization creators to create collaborative visualizations that address the specific collaboration needs of the problem. Allowing visualization creators to specify this information in accessible HTML attributes lowers the barrier for the customization. More custom event synchronization examples, such as "Where's the Square", are available on our website.

[3]Collaborative Lasso: bl.ocks.org/994a1ab12de6fb4bf21ee5c7a2461466
[4]Collaborative Brush: bl.ocks.org/b2d66e94bf90016cb285ebc9515ebc0a
[5]Where's the Square: bl.ocks.org/8f83c41f08721bd6e1d780384d9faa32

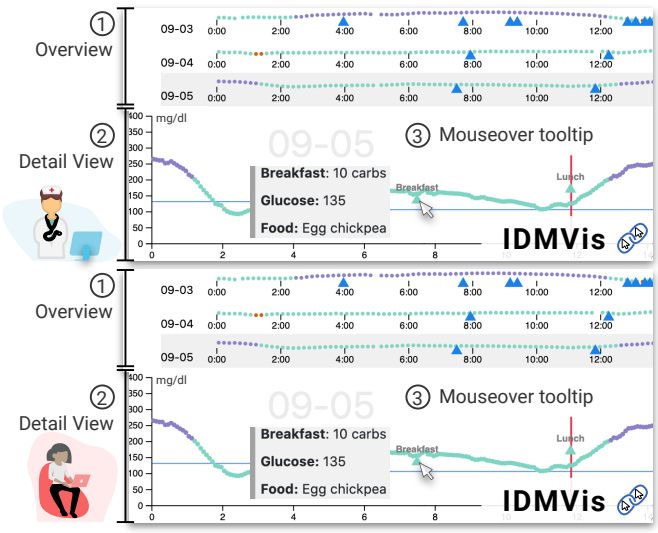


Fig. 9. A collaborative scenario where a doctor (top) and a patient (bottom) use IDMVis to share decision-making processes in diabetes management with the help of VisConnect. After the doctor selects a date from the Overview 1, both the doctor and the patient see the same graph in the Detail View 2. Upon interaction, both can see the same details on demand in the Mouseover tooltip 3.

6 Use Cases

To demonstrate the value and utility of VisConnect, we present three real prototype domain applications examples across telemedicine, remote collaborative work, and education.

6.1 Shared Decision-Making in Telemedicine

For this case study, we collaborated with authors of the recently-published IDMVis [48] diabetes visualization system to extend their work. We familiarized ourselves with literature on current diabetes management practices, and collaboration between diabetes clinicians and patients (e.g., [46, 48]). Diabetes clinicians typically communicate with patients to jointly reconstruct past events (e.g., meals, exercise, sleep, sickness). For example, clinicians identify high blood glucose data points, and investigate possible causes for hyperglycemia based on patients' memories and logs of the event times. Conventionally, this interaction happens unidirectionally as clinicians drive the conversation by identifying important data points. This type of collaboration could be partially supported by existing systems such as PolyChrome, as it supports copying the doctor's view to a patient. But this type of collaboration misses the opportunity for patients to respond to inquiries

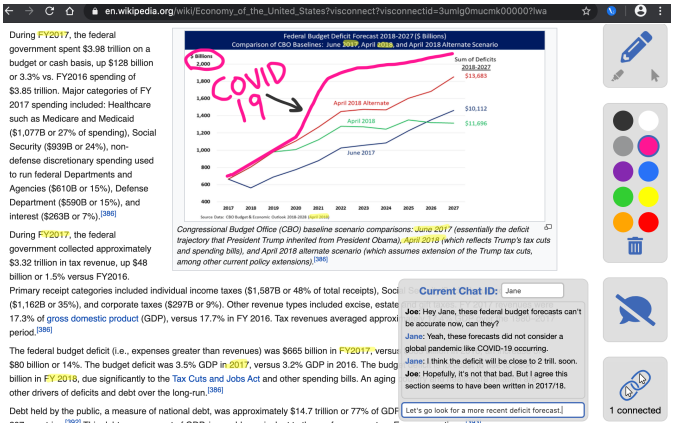


Fig. 10. A scenario where two collaborators are using Live Website Annotate to scrutinize a graph on Wikipedia. Live Website Annotate provides three main modes: Annotate (currently active)–allows annotating with a solid line, Highlight–allows highlighting with a opaque thicker line, and Interact–allows interaction with the underlying web page. A text-based chat is also available to communicate with other connected collaborators.

about the data by pointing out other relevant data in the visualization, and drawing the attention of the clinician to potentially important information. The ability for a patient to interact and have a bidirectional conversation is even more challenging in a telemedicine situation, and this two-way interaction is not supported by existing systems.

With VisConnect, on the other hand, patients can not only follow the clinicians but also redirect the conversation by initiating interaction on their own screen. This type of interaction is displayed in Fig. 9 with our IDMVis prototype. A doctor and a patient see the same temporal blood glucose data in the Overview (1) and can change alignment settings to explore blood glucose trends and patterns across days. From the overview, both the doctor and the patient can select a day of interest to reason about the change of blood glucose levels during a particular day. Upon selection, more detailed information appears in the Detail View (2). They both can hover over specific elements of the graph to examine details on demand via a mouseover tooltip (3), which are shared with their collaborator. This type of bidirectional interaction and communication has the potential to increase patient engagement and shed light on potentially important health information to the clinician.

To make IDMVis work with VisConnect, the only action taken was to add the script tag to load VisConnect in the HTML header. After this script tag was added, connected collaborators are immediately enabled to jointly decide which days and events to investigate in detail.[6] This real domain application demonstrates the ease of use of VisConnect to enable remote collaboration in existing web-based visualization tools, and the potential for VisConnect's utility to support telemedicine.

6.2 Real-time Collaborative Annotation

Shared annotations are an ongoing research effort to tackle the challenges of communication and coordination during remote collaboration on visualizations [9,21]. Collaborative annotation functionalities enable group examinations of visualization and support collective and collaborative sensemaking [41], i.e., people actively pursue shared goals and group interpretations [9]. For example, two collaborators might want to jointly learn about the U.S. economy from Wikipedia, and point out interesting facts in the text and included graphs. To illustrate VisConnect's capabilities in synchronized collaborative sensemaking, we created a Google Chrome extension, "Live Website Annotate" (LWA), which enables multiple users to simultaneously annotate existing online visualizations and websites.[7] To enable collaboration, users can install the LWA extension from the Chrome Web Store.[8] One collaborator can invite other collaborators to a shared session by creating an invitation link for the current web page. Other collaborators can join the session by pasting the link into the extension's menu. To enable annotation, the extension adds two stacked SVG elements to a website. The first SVG is used for the user interface that includes drawing options. The second SVG is used as the canvas for drawing annotations.[9] To allow the existing drawing block to be used as a collaborative annotation tool, the typical process for using VisConnect is applied: Load the VisConnect script into the current page, set the `collaboration` attribute to `live` for the website's body tag, and replace `d3.drag` with `vc.drag`. We include some other code changes to add a user interface, as shown in Fig. 10. To annotate, users can draw and highlight in ten different colors. LWA includes a chat (see Sec. 5.2) to enable communication.

This example shows how VisConnect's modular structure and compatible implementation lead to many use cases, such as annotating independently-created websites and visualizations. This is not possible with previous collaboration systems.

6.3 E-Learning for Visualization

E-learning is an increasingly popular method of learning. Some challenges of e-learning include lower levels of engagement and limited capabilities for exploration and collaboration [25, 38]. VisConnect can facilitate e-learning and visualization education: we created an interactive online exercise to help students learn how to characterize network graphs. The activity familiarizes learners with planar graphs, i.e., graphs that are possible to draw without any edge intersections [36]. The online exercise, *Planarity Party*, asks group members to collaboratively solve a puzzle. A copy of our lesson plan, including links to relevant online resources, is available in the Supplemental Material.

Protocol: This online exercise was developed for a small group setting (3–4 students per group) in a graduate-level information visualization course (CS 7250, Spring 2020, Northeastern University). First, each group selected one member to be the leader who initiated the collaborative game. Upon opening the puzzle, group leaders invited their group members to the collaboration session. Once connected, each group started the game by dragging nodes until they found a representation of the graph without edge crossings. When all edge crossings were removed, the students were informed that they solved the puzzle and moved to the next new "level". An example of four users collaborating on one such graph is included in the Supplemental Video. Every new graph was harder than the previous one, with one node added per new level. At the end of the game, each group was able to take a screenshot of the final level to see how far they got within 15 minutes. Finally, students were asked to complete a short anonymous user experience survey regarding their experience using Planarity Party, focusing on what they liked and disliked about the tool.

Results: Overall, most students found the activity enjoyable and fun, and stated that it had helped them understand concepts in visualization. For example, one student commented that *"I really liked how nodes and edges are moving in a distributed manner."* Another student mentioned how this online collaborative game helped them learn about graphs in an interesting way: *"The tool is quite interesting in the way that we can drag and drop to see how we can display nodes and links without [having] too many crosses. It is quite easy to understand the tool and designed to be easy to use."* However, students also requested some additional features. Multiple students asked to be able to save the progress in the game so it can be resumed after reloading the page. A point of disagreement among the responses was about coordinating among team members. Some people thought collaboration was easy: *"I think the tool was really fun to play around with. Collaborating was easy and the levels were challenging but still beatable."* Others thought that coordinating was difficult: *"when we are solving this as a group remotely, it is hard to discuss our ideas and other team members may ruin the idea that I had in my mind by moving nodes."* Feedback from

[6] A live demo is available at michaschwab.github.io/IDMVis/public/ and a demo video is provided in the Supplemental Material.

[7] Source code is available at github.com/shash678/Live-Website-Annotate

[8] The extension is available at chrome.google.com/webstore/detail/live-website-annotate/njhclbnmjgcghngbbbacndfcnbhgomac

[9] Drawing based on the existing block bl.ocks.org/d79632a53187f8e92b15

the instructor (an author of this paper) was also positive: "*Planarity Party* was a fun and creative way to teach the basics of laying out node-link network visualizations, and I greatly appreciated having a social and interactive exercise to use after our shift to distance learning."

Such interaction with simultaneous graph edits is not possible with previous systems (e.g., PolyChrome) as they do not automatically resolve conflicts among collaborators. Our proof of concept shows that VisConnect in e-learning can help students engage with class material and encourage learners to interact with each other's ideas.

7 Discussion

In this section, we discuss the lessons learned from this work, point out the strengths and weaknesses of our approach, and identify opportunities for future collaborative visualization research.

Direct Manipulation: VisConnect's lock system helps mitigate conflicts by blocking collaborators from simultaneously dispatching events on the same element. This model is a good fit for visualizations that modify data through direct interactions, such as drag. In visualizations that modify data through multiple elements, such as multiple input fields, VisConnect's partitioning system does not disallow simultaneous edits because the events are dispatched on different elements. If different input fields make changes to independent data, then this is not an issue. However, if multiple input fields were to change the same data attribute, then VisConnect's allowance of simultaneous interaction on these input elements could cause conflicting values in the data. To increase compatibility with non-direct data manipulations, we hope to support customizable partitioning strategies in the future.

Peer To Peer: VisConnect uses a peer to peer architecture as opposed to a centralized approach. As with all distributed systems, the main benefit is independence from a server and the main drawbacks are increased need for communication and required coordination among clients and a leader. With this architecture, information is only shared with invited collaborators and, given secure communication, no outside person can observe or modify the operations. This approach can build user trust. Independence from a central server also leads to increased **reliability**: VisConnect can be used even if the project were to be discontinued and the VisConnect server shut down.

VisConnect is currently limited to run the lock service on the collaborator that starts the session. This prevents further progress on the collaboration once the host disconnects. Improvements to VisConnect will allow the system to recognize host disconnects and transfer leadership to another client, as is typical in PAXOS leadership elections.

Single-User Assumptions: VisConnect comes with solutions to address some of the single-user assumptions in data visualizations. We provide `vc.random` as an alternative to `Math.random`, and provide collaborative interaction methods to support collaborative visualization development. These cover many common use cases, including all interaction methods listed as part of D3.js' API as well as all our examples on visconnect.us. These examples demonstrate that **our approach scales** to many different visualizations and interaction techniques. That said, visualizations can be infinitely varied, including complex aggregation of pointer events to interpret interactions, and some of these implementations are not compatible with collaborative use without change. In this respect, a combined effort is required by both the visualization community as well as by the VisConnect team to identify single-user assumptions in their implementations and to provide collaboration-ready alternatives of commonly used functions.

Collaborative Interactions: Collaboration in data visualization provides an opportunity to research interaction techniques. To address single-cursor assumptions in common interactions, VisConnect provides alternatives such as `vc.drag`, `vc.brush`, and `vc.lasso`. However, even when recreating these relatively standard interactions for collaborative use, we discovered questions about how single-user interactions should scale to collaborative use. In Sec. 5.1, we describe the question of whether collaborative brush should use a single brush for all collaborators, or if and how multiple brushes should be used and combined to work toward a common goal. The **design space for distributed collaborative interaction** needs to be characterized in future research; VisConnect is helpful for discovering and exploring this design space. As a starting point, we identify an important characteristic of collaborative interactions: whether the interaction is designed to lead to a common visualization state or different states across collaborators. For example, a "divide-and-conquer" strategy requires selecting and working on separate subsets of the data, which requires separate visualization states. In other cases, closer collaboration and coordination is required as part of a "all together" strategy, and collaborators work on the same data with the same visualization state. For example, multiple doctors might analyze a medical image and discuss the case together. In these cases, a second characteristic of the collaborative interaction is important: how are different interactions combined to lead to a shared common state? For selections, options include union and intersection of the selected subsets. More broadly, strategies to combine different interactions into a shared state should be identified to characterize collaborative interactions.

Collaborative-First Interaction Techniques: Beyond extending single-user interaction techniques for collaborative use, we hope that the community will also consider entirely **new collaborative interaction techniques.** For example, cursors could be linked and connected to form "live" 2D selection techniques, as opposed to conventional brushing techniques that require time to define start and end points. As a proof-of-concept example, we include a 2-player "Fire Fighters" game in our Supplemental Video and online[10], where two players move the ends of a safety net to catch falling people while avoiding debris that could damage the net. As with all collaboration, trade-offs exist between the need for coordination among team members and the benefits of working together. These trade-offs will be explored in future work.

Human Communication: VisConnect automates system-level communication across clients to establish the events that occur on each element. An important aspect to collaboration that is missing from this approach is the person-to-person communication. This was highlighted in the in-class study of our education use case (Sec. 6.3) in which some students expressed frustration that other collaborators had "ruined their idea" with their changes. We recognize that communication and coordination is important for collaboration and that, in many scenarios, an aligned effort is impossible without voice or other means. We show one way to mitigate this problem with live chat in our annotation tool (Sec. 6.2), and provide an example of non-verbal communication in our collaborative brush interaction, where collaborators can see each other's region of interest. Future collaborative systems should consider voice or video communication, as well as more techniques and strategies to communicate non-verbally.

8 Conclusion

We have contributed VisConnect: a system for distributed collaborative visualization that requires little to no changes to existing visualization implementations, supports most web-based SVG data visualizations, scales to many collaborators with simultaneous interactions, and balances system safety with liveness for a smooth user experience. Using an element-level lock service, VisConnect establishes consensus among clients on the sequence of executed events. With our collaborative drag, brush and lasso interactions, we make progress on the exploration of the design space of distributed collaborative interaction techniques. Through our use cases, we demonstrate the ease of implementation and use of VisConnect in multiple domains. We hope VisConnect will enable users to more effectively collaborate on visualizations; we encourage the visualization and human-computer interaction community to invest in distributed collaboration research by creating more collaborative applications and by studying interaction techniques and visualization designs specifically designed for collaborative use.

Acknowledgments

We thank Sara Di Bartolomeo, Lulu Liu and Laura South for pilot testing many of VisConnect's collaborative features and providing constructive feedback. This work was supported by the Khoury College of Computer Sciences at Northeastern University.

[10] See bl.ocks.org/dsaffo/raw/489cabbb7a6ae4c26f9bdd5b473e8d5b.

REFERENCES

[1] B. Alpern and F. B. Schneider. Recognizing safety and liveness. *Distributed computing*, 2(3):117–126, 1987. doi: 10.1007/BF01782772

[2] V. Anupam, C. Bajaj, D. Schikore, and M. Schikore. Distributed and collaborative visualization. *Computer*, 27(7):37–43, July 1994. doi: 10.1109/2.299409

[3] S. K. Badam and N. Elmqvist. PolyChrome: A Cross-Device Framework for Collaborative Web Visualization. In *Proceedings of the Ninth ACM International Conference on Interactive Tabletops and Surfaces*, ITS '14, pp. 109–118. Association for Computing Machinery, New York, NY, USA, 2014. doi: 10.1145/2669485.2669518

[4] S. K. Badam and N. Elmqvist. Visfer: Camera-based visual data transfer for cross-device visualization. *Information Visualization*, 18(1):68–93, 2019. doi: 10.1177/1473871617725907

[5] S. K. Badam, A. Mathisen, R. Rädle, C. N. Klokmose, and N. Elmqvist. Vistrates: A component model for ubiquitous analytics. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):586–596, 2019. doi: 10.1109/TVCG.2018.2865144

[6] C. Bajaj and S. Cutchin. Web based collaborative visualization of distributed and parallel simulation. In *Proceedings of the 1999 IEEE Symposium on Parallel Visualization and Graphics*, PVGS '99, pp. 47–54. IEEE Computer Society, Washington, DC, USA, 1999. doi: 10.1145/328712.319336

[7] M. Billinghurst, M. Cordeil, A. Bezerianos, and T. Margolis. *Collaborative Immersive Analytics*, pp. 221–257. Springer International Publishing, Cham, 2018. doi: 10.1007/978-3-030-01388-2_8

[8] M. Bostock, V. Ogievetsky, and J. Heer. D³: Data-Driven Documents. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2301–2309, 2011. doi: 10.1109/TVCG.2011.185

[9] S. Bresciani and M. J. Eppler. The benefits of synchronous collaborative information visualization: Evidence from an experimental evaluation. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1073–1080, 2009. doi: 10.1109/TVCG.2009.188

[10] K. W. Brodlie, D. A. Duce, J. R. Gallop, J. P. R. B. Walton, and J. D. Wood. Distributed and collaborative visualization. *Computer Graphics Forum*, 23(2):223–251, 2004. doi: 10.1111/j.1467-8659.2004.00754.x

[11] M. Bu and E. Zhang. PeerJS. `http://peerjs.com`. Accessed: 2019-04-29.

[12] H. Chung and C. North. Savil: Cross-display visual links for sensemaking in display ecologies. *Personal Ubiquitous Comput.*, 22(2):409–431, Apr. 2018. doi: 10.1007/s00779-017-1091-4

[13] G. V. Cormack. A counterexample to the distributed operational transform and a corrected algorithm for point-to-point communication. *University of Waterloo Technical Report*, 1995.

[14] C. Donalek, S. G. Djorgovski, A. Cioc, A. Wang, J. Zhang, E. Lawler, S. Yeh, A. Mahabal, M. Graham, A. Drake, S. Davidoff, J. S. Norris, and G. Longo. Immersive and collaborative data visualization using virtual reality platforms. In *2014 IEEE International Conference on Big Data (Big Data)*, pp. 609–614, 2014. doi: 10.1109/BigData.2014.7004282

[15] C. A. Ellis and S. J. Gibbs. Concurrency control in groupware systems. In *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data*, SIGMOD '89, pp. 399–407. Association for Computing Machinery, New York, NY, USA, 1989. doi: 10.1145/67544.66963

[16] Google. Google Docs. `http://docs.google.com`. Accessed: 2019-04.

[17] Google. Web fundamentals: Rendering performance. `https://developers.google.com/web/fundamentals/performance/rendering/`. Accessed: 2019-03-29.

[18] R. D. Hill. The abstraction-link-view paradigm: Using constraints to connect user interfaces to applications. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '92, pp. 335–342. Association for Computing Machinery, New York, NY, USA, 1992. doi: 10.1145/142750.142828

[19] T. Horak, A. Mathisen, C. N. Klokmose, R. Dachselt, and N. Elmqvist. Vistribute: Distributing interactive visualizations in dynamic multi-device setups. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19. Association for Computing Machinery, New York, NY, USA, 2019. doi: 10.1145/3290605.3300846

[20] P. Isenberg and S. Carpendale. Interactive tree comparison for co-located collaborative information visualization. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1232–1239, Nov 2007. doi: 10.1109/TVCG.2007.70568

[21] P. Isenberg, N. Elmqvist, J. Scholtz, D. Cernea, K.-L. Ma, and H. Hagen. Collaborative visualization: Definition, challenges, and research agenda. *Information Visualization*, 10(4):310–326, 2011. doi: 10.1177/1473871611412817

[22] P. Isenberg and D. Fisher. Collaborative brushing and linking for co-located visual analytics of document collections. *Computer Graphics Forum*, 28(3):1031–1038, 2009. doi: 10.1111/j.1467-8659.2009.01444.x

[23] K. Kim, W. Javed, C. Williams, N. Elmqvist, and P. Irani. Hugin: A Framework for Awareness and Coordination in Mixed-Presence Collaborative Information Visualization. In *ACM International Conference on Interactive Tabletops and Surfaces*, ITS '10, pp. 231–240. New York, NY, USA, 2010. doi: 10.1145/1936652.1936694

[24] C. N. Klokmose, J. R. Eagan, S. Baader, W. Mackay, and M. Beaudouin-Lafon. Webstrates: Shareable dynamic media. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software and Technology*, UIST '15, pp. 280–290. Association for Computing Machinery, New York, NY, USA, 2015. doi: 10.1145/2807442.2807446

[25] R. Kop and A. Hill. Connectivism: Learning theory of the future or vestige of the past? *The International Review of Research in Open and Distributed Learning*, 9(3), 2008. doi: 10.19173/irrodl.v9i3.523

[26] L. Lamport. Proving the Correctness of Multiprocess Programs. *IEEE Transactions on Software Engineering*, 3(2):125–143, 1977. doi: 10.1109/TSE.1977.229904

[27] L. Lamport. The part-time parliament. *ACM Transactions on Computer Systems*, 16(2):133–169, May 1998. doi: 10.1145/279227.279229

[28] R. Langner, T. Horak, and R. Dachselt. VisTiles: Coordinating and Combining Co-located Mobile Devices for Visual Data Exploration. *IEEE Transactions on Visualization and Computer Graphics*, 24:626–636, 2018. doi: 10.1109/TVCG.2017.2744019

[29] J. Leigh, A. E. Johnson, M. Brown, D. J. Sandin, and T. A. DeFanti. Visualization in teleimmersive environments. *Computer*, 32(12):66–73, 1999. doi: 10.1109/2.809253

[30] Z. Li, M. Annett, K. Hinckley, and D. Wigdor. Smac: A simplified model of attention and capture in multi-device desk-centric environments. *Proc. ACM Hum.-Comput. Interact.*, 3(EICS), June 2019. doi: 10.1145/3300961

[31] X. Liu, W. Lin, A. Pan, and Z. Zhang. WiDS Checker: Combating Bugs in Distributed Systems. In *Proceedings of the 4th USENIX Conference on Networked Systems Design & Implementation*, NSDI'07, p. 19. USENIX Association, USA, 2007. doi: 10.5555/1973430.1973449

[32] J. A. McDonald, W. Stuetzle, and A. Buja. Painting multiple views of complex objects. In *Proceedings of the European Conference on Object-Oriented Programming on Object-Oriented Programming Systems, Languages, and Applications*, OOPSLA/ECOOP '90, pp. 245–257. Association for Computing Machinery, New York, NY, USA, 1990. doi: 10.1145/97945.97975

[33] R. Mettälä, J. Piispanen, M. Sahinoja, and A. Sutinen. Data synchronization, Aug. 4 2009. US Patent 7,570,668.

[34] S. North, C. Scheidegger, S. Urbanek, and G. Woodhull. Collaborative visual analysis with rcloud. In *2015 IEEE Conference on Visual Analytics Science and Technology (VAST)*, pp. 25–32, Oct 2015. doi: 10.1109/VAST.2015.7347627

[35] T. Pattison and M. Phillips. View Coordination Architecture for Information Visualisation. In *Proceedings of the 2001 Asia-Pacific Symposium on Information Visualisation - Volume 9*, APVis '01, pp. 165–169. Australian Computer Society, Inc., AUS, 2001. doi: 10.5555/564040.564061

[36] B. Plestenjak. An algorithm for drawing planar graphs. *Software: Practice and Experience*, 29(11):973–984, 1999. doi: 10.1002/(SICI)1097-024X(199909)29:11<973::AID-SPE268>3.0.CO;2-B

[37] J. C. Roberts. State of the Art: Coordinated Multiple Views in Exploratory Visualization. In *Fifth International Conference on Coordinated and Multiple Views in Exploratory Visualization (CMV 2007)*, pp. 61–71, 2007. doi: 10.1109/CMV.2007.20

[38] I. Roffe. E-learning: engagement, enhancement and execution. *Quality assurance in education*, 2002. doi: 10.1108/09684880210416102

[39] C. Sun and C. Ellis. Operational transformation in real-time group editors: Issues, algorithms, and achievements. In *Proceedings of the 1998 ACM Conference on Computer Supported Cooperative Work*, CSCW '98, pp. 59–68. Association for Computing Machinery, New York, NY, USA, 1998. doi: 10.1145/289444.289469

[40] C. Sun, X. Jia, Y. Zhang, Y. Yang, and D. Chen. Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems. *ACM Trans. Comput.-Hum. Interact.*, 5(1):63–108, Mar. 1998. doi: 10.1145/274444.274447

[41] J. J. Thomas. *Illuminating the path: The Research and Development Agenda for Visual Analytics*. IEEE Computer Society, 2005.

[42] J. J. Thomas and K. A. Cook. *Illuminating the Path: The Research and Development Agenda for Visual Analytics*. National Visualization and Analytics Center, 2005.

[43] C. Thum and M. Schwind. Synchronite - A Service for Real-Time Lightweight Collaboration. In *2010 International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*, pp. 215–221. IEEE, 2010. doi: 10.1109/3PGCIC.2010.36

[44] D. Wigdor, H. Jiang, C. Forlines, M. Borkin, and C. Shen. Wespace: The design development and deployment of a walk-up and share multi-surface visual collaboration system. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '09, pp. 1237–1246. Association for Computing Machinery, New York, NY, USA, 2009. doi: 10.1145/1518701.1518886

[45] D. Wigdor, H. Jiang, C. Forlines, M. Borkin, and C. Shen. Wespace: The design development and deployment of a walk-up and share multi-surface visual collaboration system. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '09, pp. 1237–1246. Association for Computing Machinery, New York, NY, USA, 2009. doi: 10.1145/1518701.1518886

[46] J. I. Wolfsdorf. *Intensive Diabetes Management*. American Diabetes Association, fifth edition ed., 2012. doi: 10.2337/9781580404587

[47] E. K. K. Yasojima, B. S. Meiguins, and A. S. Meiguins. Collaborative augmented reality application for information visualization support. In *2011 15th International Conference on Information Visualisation*, pp. 170–175, 2011. doi: 10.1109/IV.2011.44

[48] Y. Zhang, K. Chanana, and C. Dunne. IDMVis: Temporal Event Sequence Visualization for Type 1 Diabetes Treatment Decision Support. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):512–522, Jan 2019. doi: 10.1109/TVCG.2018.2865076