7

INNOVATION IN CONTENT ANALYSIS

Freezing the flow of liquid news

Rodrigo Zamith

As scholars of journalism take greater interest in analyzing online news content, they find themselves facing novel methodological challenges (Karlsson and Sjøvaag, 2016). One key challenge involves dealing with the increasingly "liquid" nature of that content and its ability to rapidly mutate and sometimes disappear (Karlsson, 2012a; Sjøvaag and Stavelin, 2012). Indeed, such developments have already begun to complicate the notion that such content "can be considered as a finished, static object of study" (Deuze, 2008: 861)—a key consideration for content analyses.

This challenge becomes even more pressing as an increasing amount of news content is produced and distributed through networked, digital platforms, developed for increasingly technically savvy audiences and increasingly powerful devices. This results in larger volumes of progressively more complex artifacts that leverage new technologies and can quickly change (Zamith, 2017b). That, in turn, demands the development of new methods and frameworks for effectively "freezing" content into static snapshots that may be computationally analyzed in an efficient manner in order to permit rigorous content analyses and further scholars' understanding of the evolution of news content in an increasingly liquid ecosystem (Karlsson and Sjøvaag, 2016).

This chapter focuses on evaluating a range of methods for dealing with this growing challenge. It begins by synthesizing the literature around the concept of "liquid news" and how it complicates traditional approaches to content analysis. It then describes key considerations about the technical aspects of web pages and reviews existing methods for freezing online news content. Those methods are then employed and critiqued through a case study whereby link information is extracted from different parts of the fluid and interactive *New York Times* homepage. Finally, recommendations involving common research scenarios are offered, and directions for future research are presented.

Background

Drawing on the work of Polish social theorist Zygmunt Bauman, Deuze (2008) argues that newswork must become "liquid" in order to adapt to the media ecology that began to emerge at the turn of the century. By this, Deuze means that "uncertainty, flux, change, conflict, and revolution are the permanent conditions of everyday life" (ibid.: 851) for the institution of journalism – much as it is for modern society (Bauman, 2005). These conditions apply both to newsworkers and the news content they produce.

A key insight derived from this work is that online news content has become increasingly mutable and ephemeral (Karlsson and Strömbäck, 2010; Zamith, 2016). As Karlsson and Sjøvaag (2016) argue, the boundaries of liquid journalism, if they exist at all, are distinctly permeable. For example, the text of an online news story may be updated several times with no apparent indication that any changes have been made to the original document. Indeed, a document's Uniform Resource Locator (URL) often does not change in response to modifications to that document, and it may be impossible to retrieve its previous iteration (Karlsson and Sjøvaag, 2016). Drafts therefore begin to bleed into one another. Similarly, whereas just a few variations of a news product were previously possible (e.g., a handful of location-specific editions of the morning newspaper), liquid journalism faces no such limitation. Homepages may be updated at will by humans, and personalized and algorithmically defined variants can be generated in milliseconds.

Such affordances are not only theoretically possible but are becoming increasingly important to news organizations. As scholars have observed, immediacy and interactivity are key characteristics that help distinguish online news from its analog counterparts (Boczkowski, 2004; Karlsson, 2011). Immediacy refers both to the notion that consumers can view content immediately after it is produced (Lim, 2012) and to the expectation among modern news consumers that content will be updated the next time it is refreshed (García Avilés et al., 2004). Interactivity primarily refers to individuals' ability to shape, in real time, the form and content of the material and environment they are engaging with and within (Steuer, 1992), such as opening and closing modules on a page or engaging in co-production through user comments (Martin, 2015). Scholars have also observed that multimodality and the hyperlinked nature of online news serve as distinguishing characteristics (Karlsson, 2012a).

With these considerations in mind, one may adopt Karlsson's (2012a: 387) conceptualization of liquid news as "an erratic, continuous, participatory, multimodal and interconnected process that is producing content according to journalistic principles". To help illustrate the broad nature of the concept and begin charting empirical dimensions that may be explored, Karlsson proposes eight theoretically grounded variables for examining news liquidity. These include the number of versions (i.e., drafts) a given news item goes through, the extent of user involvement through participatory features, and the "accuracy problem" that may emerge with fluid events. This chapter focuses on assessing change as a response to modifications by the producer (immediacy) and by the consumer (interactivity), as these appear to be two of the most common interests among scholars of digital journalism.

As Deuze (2008: 860) argues, news organizations should "embrace the uncertainty and complexity of the emerging new media ecology, and enjoy it for what it is: an endless resource for the generation of content and experiences by a growing number of people all around the world." Scholars should similarly embrace this shift as it can help them peer into the "black box" of journalism and better understand it as a process that may be witnessed in real time and in public (Tumber, 2001; Karlsson, 2011). Indeed, examinations of liquidity have already pointed to the time-dependent nature of immediacy and versionality on the websites of news organizations (Widholm, 2016), how distinct pictures of crises are painted through rapidly changing media framing (Karlsson, 2012b), and how the popularity of news items may affect their future prominence on a homepage (Zamith, 2016).

Liquid news and content analysis

Embracing the paradigm behind liquid journalism presents a number of challenges to content analysis methodology. As Karlsson and Sjøvaag (2016) aptly note, the logic of content analysis is largely embedded in the affordances of print media. Digital media – and online news in particular – complicate notions of space and time, making it "difficult to pin down where the

news is produced, distributed, and consumed as news appears in so many places – and comes in so many forms – at once" (Karlsson and Sjøvaag, 2016: 180). They observe that content analyses of online news – both manual and computational – require a set of considerations that distinguishes them from traditional content analyses of analog materials. These include the mode of analysis, variable design, scope, sampling procedure, unit of analysis, storage, generalizability, key agent, and the aim of the analysis (Karlsson and Sjøvaag, 2016).

The notion of time, in conjunction with the concepts of immediacy and interactivity, is particularly important in the present context. As Deuze observes:

> the study of content has always rested on the premise that content actually exists, that it genuinely can be considered as a finished, static object of study. In the current media ecology of endless remixes, mashups, and continuous edits, that is a problematic assumption.
>
> *(Deuze, 2008: 861)*

The researcher's challenge, then, is to turn a dynamic object into a static one (or a sequence of static ones). This process of "freezing" (Karlsson and Strömbäck, 2010) is necessary in a manual content analysis in order to ensure that coders are viewing the same content when an assessment of intercoder reliability is taking place. In a computational analysis, frozen content is necessary for training, tweaking, and testing algorithms, and to ensure reproducibility (Zamith and Lewis, 2015). Perhaps most importantly, for any assessment of some aspect of a dynamic object's liquidity to occur, it must be frozen into a series of static snapshots that can be used as the basis of an evaluation of that object's fluidity.

As Karlsson and Sjøvaag (2016) observe, traditional content analyses are often performed *ex post facto*, with some central repository archiving all of the content that will be studied. For example, libraries may keep an archive of all the issues of a given magazine or all recorded episodes of an evening newscast, and third-party vendors may maintain a database with all the articles appearing in a particular newspaper. Unfortunately, there is no such record of the web – even for its most popular sites. While digital archives like the Internet Archive's 'WayBack Machine' may store occasional snapshots of what the content looked like at a certain point in time, those snapshots are infrequent and unevenly spaced, presenting serious limitations to scientific analyses (Karpf, 2012). Additionally, these snapshots may be incomplete; not only might portions of the page be missing, but linked content is sometimes left unarchived.

In the absence of third-party options, researchers must often collect their own data (Karlsson and Sjøvaag, 2016; Zamith, 2017b). The upside is that such collection need not be difficult or resource-intensive and can be tailored to the research inquiry. The downside is that *ex post facto* research is often impossible. Researchers must either anticipate an event and plan ahead of time – not always a possibility with breaking news – or already be collecting data from the desired sources when an event takes place.

The structure of online web pages

The study of liquidity may focus on a variety of elements, but it often centers on change – such as the addition of some elements and the removal of others – on digital documents like web pages (Karlsson and Strömbäck, 2010; Karlsson, 2012a; Lim, 2012; Sjøvaag et al., 2016; Zamith, 2016). By their very design, web pages are highly structured documents, typically consisting of a mixture of the HyperText Markup Language (HTML), Cascading Style Sheets (CSS), and JavaScript, a dynamic and interpreted programming language that can add functionality to web pages through scripts that are executed on the user's computer. All of a page's source code is downloaded when a user requests the page from the server (i.e., by accessing the page's URL). It

is then interpreted by the user's browser, which renders the code and any associated media into the audio-visual objects the user sees and hears when the page is loaded.

The nature of the web therefore provides researchers with access to most, if not all, of the elements necessary for storing a copy of a web page locally. Additionally, all of this computer code provides some structure to the data social scientists typically want to capture. For example, the beginning of an unordered list is denoted in an HTML document by the tag `<ul>`, and its end by `</ul>`. Each bullet point within that list is then denoted by `<li>` and `</li>`, respectively, and the text of the bullet point is denoted by the text between those two tags. The position of the bullet point within that list (i.e., if it is the first or third bullet point) is denoted by the sequence of the code.

Specific elements within the page can sometimes be identified by a researcher through its `id` attribute or some unique combination involving the `class` attribute, either of which may be included by the page's author. Researchers typically do this by writing selectors – pieces of code that identify content based on the presence of specified attributes (see Sjøvaag and Stavelin, 2012). If those attributes are not set, the researcher can also use the element's place in the structure of the document, relative to other elements, to identify and extract the desired data.

Furthermore, as scholars have noted, news organizations typically use content management systems in order to organize information, automatically place it on a template, and publish it online (Diakopoulos, 2017). This is notable because it means the structure of each news organization's homepage will generally only contain minor variations over extended periods of time. As Zamith (2017b) found in his analysis of the homepages of 21 news organizations, each organization had at most a handful of distinct layouts. Thus, although the homepage was liquid over the course of the day – new stories were added and removed, promoted and demoted, and different "related content" was deemed relevant – the layout and underlying code saw limited variation. This allows the researcher to leverage the uniformity of a web page's code and identify common patterns that permit the extraction of particular data. That, in turn, allows for both the preservation of the original artifact and the extraction (and subsequent freezing) of interesting aspects of that artifact.

Methods for freezing liquid news content

A growing body of scholarship has focused on developing approaches for freezing liquid content. Karlsson and Strömbäck (2010) point to three techniques for accomplishing that. The first technique is to take a screenshot of the content, typically by using a computer program or simply utilizing the operating system's native "print screen" functionality. As they observe, screenshots are helpful because they provide a visual replica of the content at the time of viewing. Image files like JPEGs and PNGs should not change if the snapshot is opened with different software (or versions of those software). However, they observed that screenshots typically only capture the content appearing in a specific frame. That is, any content that requires the user to scroll up or down to become visible would not be captured (unless the researcher goes through the messy process of taking multiple screenshots and stitching them together). While this is true of most software, Widholm (2016) and Zamith (2017b) have pointed to solutions that are able to capture the entire page in a single image. Nevertheless, as Zamith (2017b) observes, valuable information is lost when interactive objects are transformed into flat images (e.g., a JPEG), such as the URL a headline links to.

The second technique noted by Karlsson and Strömbäck (2010) is to print a copy of the content, often to a PDF file. As they observe, this often triggers a "print version" of the page, which may omit valuable data such as user comments, interactive visualizations, and "related content". When a print version is not available, the browser will typically attempt to resize the content to

fit different dimensions (e.g., letter-sized paper using a portrait orientation). This requires the elements on a page to be automatically rearranged, often resulting in copies that look very different from the original. Additionally, as with screenshots, a great deal of valuable information is lost in the conversion.

The third technique noted by Karlsson and Strömbäck (2010) is mirroring the page. This involves downloading all the computer code necessary to render the page, including all of the associated media (e.g., images and stylesheets). One may do this manually by using features in modern web browsers (i.e., "save page as") or computationally using software like Wget and HTTrack (e.g., Sjøvaag and Stavelin, 2012; Lewis et al., 2013). The advantage of using this technique is that it preserves the greatest amount of information, such as the specific styling options for a particular element and its structural position within the hierarchy of the page. However, tools like Wget and HTTrack often fail to capture all of the information necessary to generate an exact replica of a page at the time it was mirrored. Specifically, they often fail to capture and relink (to a local copy) certain elements required to accurately display that page, such as JavaScript files. Moreover, they are unable to process JavaScript actions for loading external content (e.g., loading user comments that require the user to click on a button titled "view comments").

Recently, Karlsson and Sjøvaag (2016) called for the development and assessment of computational approaches for handling liquid content in response to the growing and rapid shift toward digital media production and consumption. Such methods are typically preferred because they enable large-scale analyses, offer perfect reliability, permit post hoc malleability, and can improve transparency (Zamith and Lewis, 2015). Recent scholarship has responded to that call with the development of more advanced methods capable of capturing both the potential for rapid change as well as the more advanced, interactive features of websites (e.g., Widholm, 2016; Zamith, 2017b). For example, Zamith (2017b) has pointed to the value of the Selenium framework for automating traditional web browsers. This permits the processing of JavaScript and storing a complete rendering of a web page through full-page screenshots as well as the source code after it has been processed by the browser. Moreover, it allows the researcher to simulate user behavior, such as scrolling down the page and clicking on particular content in order to expand it (Zamith, 2016). However, Selenium often comes at a considerable computational cost in terms of the processing power and memory required, which may present a challenge to truly large-scale analyses (Zamith, 2017b). One underexplored alternative, Zamith (2017b) suggests, is the use of web browsing engines that don't require a graphical user interface to be loaded (headless browsing).

The array of potential solutions and limitations therefore begs the question: Which solution is best for freezing the flow of liquid news?

Case study

In order to illustrate the effectiveness and limitations of each of those methods, the aforementioned tools and approaches were used to freeze the homepage of the *New York Times* on April 19, 2017. This examination was designed to attempt to replicate, on a micro scale, the data collection performed by Zamith (2016). In that study, Zamith froze the homepages of 14 news organizations every 15 minutes over two months. Of particular importance were link data pertaining to the news items that appeared in areas of the page deemed to be prominent, the news items that appeared on the list of most-viewed items on the homepage, and the news items that appeared elsewhere on the page. Those data were then used to assess the impact of an item's popularity on its subsequent prominence and likelihood of being removed from the homepage.

This examination was performed using free, open-source software on consumer-grade computer hardware.[1] A copy of the source code used as part of this case study may be downloaded from the author's website, www.rodrigozamith.com.

Manually saving content

The first option for freezing content is also the simplest: fire up a browser, surf to the *Times*' homepage, and save the content. As noted by Karlsson and Strömbäck (2010), there are two primary ways to manually save content: saving its source code and creating a visual replica.

Saving a page's source code simply requires making use of the "Save Page As" feature of the browser. That feature includes options to save an "HTML only" copy of the page as well as a "complete" one. The first created a single file comprised of the page's HTML source code, which included information about the structure of the page and much of its text content. However, a great deal of the content on the page was simply referenced by the HTML source code (and thus not embedded in it). For example, the pictures associated with the top headlines were hosted on the *Times*' server – which may change or be removed. If the researcher is merely interested in the headlines and blurbs, this feature may suffice. However, if a complete recreation of the page is necessary, this approach may prove problematic.

The second option, storing a "complete" copy of a page, proves more versatile. It not only creates a copy of the source code but also of all the assets loaded by the browser (e.g., image files and stylesheets).[2] Notably, the HTML source code is automatically modified to reference the locally stored copies. Additionally, the "complete" copy of the page also stores any modifications to the page as a result of the execution of JavaScript at the time of the snapshot. For example, the "complete" copy was able to store the code necessary to recreate the JavaScript-powered list of most-viewed items, if the user scrolled down to it. The "HTML only" copy did not. Nevertheless, some aspects of the page may not be fully replicated. Areas containing Flash video files, for example, were left as blank spots on the "complete" copy. As one might expect, the "complete" copy was considerably larger (16.4 megabytes) than the "HTML only" copy (0.21 megabytes) in this case study, which could present issues in a large-scale analysis.

Creating a visual replica of the page is usually done through one of two options: saving a PDF copy of the page or taking a screenshot (Karlsson and Strömbäck, 2010). While Mozilla's Firefox did not have built-in functionality for creating PDFs, most modern operating systems offer a virtual printer by default. This operation typically requires the page to be reformatted to fit the standard paper size (i.e., letter) and some websites employ custom stylesheets that remove certain elements and reflow the text. While this may prove to be sufficient if the research inquiry focuses solely on the text of the page – provided the reformatted version does not omit it – the resulting PDF would not be an actual replica of what is observed on the screen. In the case of the *New York Times*' homepage, no such stylesheet was offered. Instead, the browser attempted to reformat content itself, resulting in a 10-page document with multiple overlapping objects that looked very different from the rendered web page (and was difficult to read).

The second option is to take a screenshot of the page. Prior work has referenced the use of software, such as an operating system's native "print screen" tool, to capture such images, and lamented the fact that such software is typically only able to capture the portion of the page that is visible on a screen at any given moment. However, Mozilla's Firefox includes a "Developer Toolbar" feature that allows the program to generate a single image that encompasses the entire page. This is done by entering the command, `screenshot –fullpage [filename]`, which in turn generates a PNG image file. This effectively bypasses the limitations previously observed in the literature, though the resulting images are quite sizable. A single image from this case study was a hefty 6.3 megabytes.

While saving "complete" copies of the *Times*' homepage would be theoretically sufficient for freezing the data necessary to perform Zamith's (2016) analysis, it would be impractical. It would require an individual to simultaneously load 14 pages every 15 minutes and perform a series of actions on each page. Over the course of two months, this would add up to nearly 82,000

snapshots that would need to be stored by a human being around the clock – introducing great potential for human error.

Content retrieval software

In order to automate the freezing process, many researchers have turned to content retrieval software, such as Wget, HTTrack, and the Requests library for Python. The first of these tools, Wget, will simply download and store a copy of the contents of the URL provided. For example, it is possible to store the equivalent of an "HTML only" copy of the *New York Times*' homepage through the simple command `wget -O 'nyt.html' 'www.nytimes.com'`. Researchers may run Wget on an automated timer (e.g., every 15 minutes) by using system-provided tools, such as Linux's Crontab. Because only the original HTML source code is downloaded, the aforementioned limitations apply.[3]

The second tool, HTTrack, is designed to create a mirror (an exact copy) of a website. HTTrack offers functionality that is similar to storing a "complete" copy of a page through Firefox, as well as more advanced features. Like a "complete" copy, all of the elements linked to from a starting point (i.e., the *Times*' homepage) are automatically downloaded, and references to that external content are rewritten to refer to the new, local copies. HTTrack also offers the ability to crawl websites through recursive downloading. For example, the command, `httrack 'www.nytimes.com' -O 'nyt' -mirrorlinks -r2` instructs HTTrack to not only download the *Times*' homepage but also anything linked to from that page, such as the news stories that appear on it.[4] Expanding the scope of the mirror quickly increases the size of the contents, though HTTrack will reuse assets when possible to conserve space.

The third tool is a library for the popular Python programming language called Requests. Python is a general-purpose programming language used by several computational social scientists (e.g., Sjøvaag and Stavelin, 2012; Zamith, 2017b) due to its blend of power and ease of use. It also includes a number of libraries that make it easier for the researcher to perform common functions, from accessing the Twitter API (Tweepy) to creating graphical plots (Matplotlib). Requests allows the researcher to embed Wget-like functionality into a Python script. For example, the source code from the *Times*' homepage can be obtained with the following code: `page_source = requests.get('www.nytimes.com').text`. This is useful when one wants to collect a particular subset of data from a broader document. For example, the BeautifulSoup library for Python – which transforms HTML code into a document that can be computationally navigated – can be paired with Requests to identify and download all of the links appearing in a certain part of the website. One could then create a variable with the navigable source code with: `page_source_soup = BeautifulSoup(page_source, "lxml")`. Then one could extract all of the links appearing near the top portion of the page (identified in the page's source code by the `id` attribute `top-news`) with: `[x['href'] for x in page_source_soup.find('section', {'id': 'top-news'}).find_all('a', href=True)]`. Finally, the Requests library can be leveraged to download each of those links, using code similar to that here.

Although these tools range in difficulty, they all enable automation of some sort. This permits the researcher to engage in unsupervised data collection, perhaps checking in periodically to confirm the operations are running as expected. However, none of these tools are able to fully process JavaScript code, which restricts their utility within the modern web. In the present case study, none were able to load the list of most-viewed items because of that limitation – therefore precluding access to a key set of data. Moreover, these tools are unable to generate visual replicas of the content (i.e., screenshots).

Automated browsing software

An emerging set of software that permits the freezing of content from more complex websites is automated browsing software, such as CasperJS and the Selenium WebDriver. These software allow computer code to simulate human actions using either a full web browser engine, like Mozilla's Gecko or KDE's WebKit, or a full browser, like Mozilla's Firefox or Google's Chrome. This, in turn, allows for the complete simulation of a regular browsing session, which includes the processing of client-side actions in JavaScript code.

One particularly useful tool for automated browsing is CasperJS, which permits headless browsing or scripted web browsing without a graphical user interface. It builds on the PhantomJS software, which in turn employs the WebKit browser engine used by Apple's Safari and Google's Chrome and thus tends to render pages as intended by the pages' authors. Because it is a headless solution, it requires relatively few resources to process a web page – though not as little as content retrieval software. CasperJS natively provides a JavaScript API to allow computer scripts to simulate user behavior and automate browsing. For example, the source code from the *New York Times*' homepage can be obtained with the following code: `casper.start('www.nytimes.com')`. A screenshot of the entire page can be taken by appending the following code: `this.capture('nyt.png')`.

Unlike the previous tools, however, CasperJS is able to process JavaScript code and interact with elements on a page. This enables it to load the list of most-viewed items on the *Times*' homepage, which requires the user to scroll down to the list of most-viewed items before it will load any content, and then click on the desired list (i.e., most-viewed versus most-emailed). To do this, one may use `this.scrollToBottom()` to scroll to the bottom of the page – thereby ensuring all position-specific elements load – and `this.click('.most-viewed-tab')` to simulate a user clicking on the list of most-viewed items, which is distinguished by the *Times*' homepage through the `class` attribute `most-viewed-tab`. Finally, CasperJS is also able to simulate waiting through the `.wait()` and `.waitForSelector()` functions, which is sometimes necessary in order to avoid interstitial ads that have become common on many news sites. Notably, CasperJS is able to save a page's HTML source code after it has been processed by the browser. This is a key distinction because it permits subsequent analysis of areas of a page that require JavaScript to be displayed with tools like Python's BeautifulSoup library. While it is possible to extract the links appearing in the list of most-viewed items using CasperJS, such a process is more cumbersome than when using a solution like BeautifulSoup.

Finally, the most comprehensive tool is the Selenium WebDriver (hereafter called Selenium), which permits the automation of complete web browsers like Firefox and Chrome.[5] Selenium is fully implemented in and supported by a number of programming languages, including Python, Ruby, and Java, making it accessible to programmers with different expertise. Using Python, for example, Firefox may be opened using `driver = webdriver.Firefox()`. Then, `driver.get("www.nytimes.com")` may be used to open the *Times*' homepage, `driver.page_source` to access its source code, and `driver.save_screenshot("nyt.png")` to save a screenshot. To select the list of most-viewed items, one can use `driver.find_element_by_class_name("most-viewed-tab").click()`.

Selenium permits the researcher to not only replicate anything that may be accomplished with CasperJS but also to use plugins (e.g., Adobe Flash), add-ons (e.g., uBlock Origin), and offline storage (e.g., for websites like the *Times*' Today's Paper web app). Additionally, in the author's experience, Selenium will typically do a better job than CasperJS at rendering very complex websites that make extensive use of JavaScript and bleeding-edge HTML and CSS features. Moreover, Selenium offers the advantage of easy, simultaneous integration of the browsing

session with libraries like BeautifulSoup. This allows the researcher to perform common actions like extracting links from areas of interest on a page and comparing pages to previous snapshots within a single script, thereby removing the need for storing multiple versions of the same document or adding steps to a workflow.

These benefits, of course, come at the expense of considerably higher computational costs. Using a full browser like Chrome consumes far more memory and CPU cycles than the headless use of the WebKit engine. Additionally, browsers like Firefox and Chrome require a graphical user environment, and thus display hardware and windowing software. There are framebuffer emulators like Xvfb for Linux, which permit Firefox to run in a terminal session (i.e., command line). While such emulators are considerably less computationally expensive than a full windowing system and desktop environment, they nevertheless require additional resources, which can be problematic when working with a large number of websites.

For the purposes of replicating Zamith's (2016) study, either of these two solutions would be suitable. However, given that CasperJS is considerably less resource-intensive, it should be favored for that kind of work. Alternatively, Selenium may be paired with PhantomJS – as opposed to a full browser – to reduce the resource load while simultaneously incorporating popular Python libraries like BeautifulSoup.

Weighing options and future directions

As this chapter has indicated, there are several potential solutions for freezing the flow of liquid news. The selection of a solution is often dependent on the researcher's technical ability, the nature of the research inquiry, and the complexity of the digital artifacts being studied. For simple analyses, manually saving pages or taking screenshots may do the trick.

However, demand for computational solutions is likely to increase since the study of the liquidity of news will increasingly require researchers to look at shorter periods of time as expectations of immediacy grow (García Avilés et al., 2004; Lim, 2012). This is an important consideration because research designs that involve short intervals tend to require computationally efficient solutions and generate large datasets. As Karlsson (2012a) notes, the dearth of empirical analyses of the liquidity of content is partly due to the difficulty of the work. Indeed, Zamith (2017a) reported having to limit the sample size of that study due to the computational and memory requirements of automating Firefox through Selenium, and Zamith's (2017b) analysis required nearly 650 gigabytes of storage space.

Computational solutions can be accessible to those with limited technical backgrounds, though. Content retrieval software like Wget and HTTrack are very accessible and are capable of freezing enough aspects of liquid content to be usable in a range of applications (e.g., Sjøvaag and Stavelin, 2012). Put differently, such software can deal with the immediacy of liquid news, though it may fail to deal with its interactive elements. Moreover, tools like Wget are computationally inexpensive and thus advantageous for large-scale analyses. It is thus advisable to try such software first before progressing to more powerful solutions.

However, for certain projects and more complex pages such as the ones in this case study more advanced solutions are required. As Martin (2015) and others have observed, news organizations are increasingly turning to integrated third-party platforms in order to incorporate important interactive features onto websites, such as reader comments and indicators of trending content. Moreover, increasingly popular features like interactive data visualizations require the use of JavaScript to simply appear on a page. Automated browsing tools are necessary when such data—or more faithful replicas—are of import to the researcher. In the present case study,

headless browsing software like CasperJS offered superior performance to a Firefox- or Chrome-powered Selenium solution while having similar functionality for dealing with the interactivity of modern websites. Researchers are thus encouraged to initially consider that option for navigating complex websites, though a full browser may be occasionally required.

An additional consideration researchers may face is whether they should store their snapshots as screenshots or source code – or both. Full-page screenshots offer the best chance that the content will look exactly the same over time and may be ideal for qualitative analyses of text and graphical content, as well as visual analyses of the assemblage of content. However, a great deal of important information is lost in that conversion (e.g., the ability to extract the text or link information), which hampers subsequent computational analyses. Moreover, screenshots are often several times larger than the page's source code – though they are typically smaller than storing a "complete" copy of a page that includes all the referenced media. In many instances, researchers would be well-served by storing both screenshots and the page's source code. Zamith (2017b) reported that screenshots served as a helpful aid for manually ensuring that content was being stored correctly and that the algorithms employed for computationally analyzing content were making valid decisions.

Methods for freezing content will need to continue to evolve in order to keep up with emerging trends in media production, distribution, and consumption. For example, recent work has found that individuals are doing more of their news consumption on mobile devices and that news organizations are devoting more resources to mobile news apps and chat apps (Belair-Gagnon et al., 2017). Future efforts should seek to explore efficient ways to freeze the content appearing on those apps, whether through official application programming interfaces (APIs) or by emulating the experience. Additionally, as some services' APIs become more restrictive (e.g., Twitter), researchers must identify solutions that can yield quality data using less-restrictive modes of access, such as by scraping the websites of those services. Finally, as algorithms become more prominent in social life – leading to questions about the development of "filter bubbles" around news consumption and political discourse – researchers must develop methods for simulating distinct user experiences in order to gather data on those algorithms' impact on content. While the automated browsing approaches described here may serve as a starting point, additional work is required for identifying the most efficient frameworks for capturing distinct experiences.

Conclusion

News is becoming increasingly "liquid" as more of it is produced, distributed, and consumed through digital platforms. Researchers have responded by developing new approaches for analyzing news content and identifying multiple tools to "freeze" it. However, such tools vary in efficiency and sophistication and thus range in their ability to capture the sorts of liquid objects that are of greatest interest to scholars.

As has been shown in this chapter, there is no single "best" tool for freezing liquid content. Old tools like Wget remain useful for less complex pages and require few computational resources. New tools like CasperJS are better able to capture interactive elements and more-intricate features of a page, though they come with higher computational costs. The appropriateness and effectiveness of a tool thus depends on the nature of the research inquiry and the objects that will be studied. However, one thing is clear: As the web evolves and news products become more complex, existing approaches to content analysis will need to be refined and new tools developed.

Further reading

A number of works helped shape this chapter and are recommended for further reading. Deuze's (2008) "The Changing Context of News Work" is a great introduction to the concept of "liquidity" and how it extends to newswork. Karlsson's (2012a) "Charting the Liquidity of Online News" and Karlsson and Strömbäck's (2010) "Freezing the Flow of Online News" do a wonderful job of applying that concept to news content and highlighting the methodological challenges it introduces to content analysis. Karlsson and Sjøvaag's (2016) "Content Analysis and Online News" aptly explicates an approach they call "liquid content analysis", and how it departs from traditional methodology along the many steps of the process. Finally, Zamith's (2017b) "Capturing and Analyzing Liquid Content" offers a detailed and practical description of a complex computational analysis of liquid content that includes useful snippets of code and explains them.

Notes

1 The following software were used: BeautifulSoup (4.5.1), CasperJS (1.1.4), HTTrack (3.49), Mozilla Firefox (50.0.2), Selenium (2.53.4), and Wget (1.15).
2 Mozilla also offers a downloadable add-on to Firefox called "Mozilla Archive Format" that aims to create an even more faithful, system-independent, and compressed copy of a page.
3 Wget includes a number of useful options to augment its basic functionality. For example, the command `wget -E -H -k -K -p -P 'nyt' 'www.nytimes.com'` would achieve a result similar to saving a "complete" copy with Firefox. By appending the arguments `-r -l 2`, it is possible to replicate the recursive functionality described for HTTrack. However, HTTrack offers more functionality than Wget for these kinds of operations.
4 The last argument in that command ensures it goes no further than the homepage. Additional arguments may be used to restrict the crawling to a small set of relevant domains.
5 The Selenium WebDriver is part of a broader framework for testing web applications, which includes a complete integrated development environment and its own programming language. It may also be used to automate CasperJS and PhantomJS for those who do not wish to write their own JavaScript. It also supports SlimerJS, which provides near-headless access to Mozilla's Gecko engine.

References

Bauman, Z. (2005) *Liquid Life*. Cambridge: Polity.

Belair-Gagnon, V., Agur, C. and Frisch, N. (2017) "The changing physical and social environment of newsgathering: A case study of foreign correspondents using chat apps during unrest." *Social Media + Society*, 3(1), 1–10.

Boczkowski, P. J. (2004) "The processes of adopting multimedia and interactivity in three online newsrooms." *Journal of Communication*, 54(2), 197–213.

Deuze, M. (2008) "The changing context of news work: Liquid journalism for a monitorial citizenry." *International Journal of Communication Systems*, 2, 848–865.

Diakopoulos, N. (2017) "Computational journalism and the emergence of news platforms." In B. Franklin and S. Eldridge (eds.), *The Routledge Companion to Digital Journalism Studies*. New York, NY: Routledge (pp. 176–184).

García Avilés, J. A., León, B., Sanders, K. and Harrison, J. (2004) "Journalists at digital television newsrooms in Britain and Spain: Workflow and multi-skilling in a competitive environment." *Journalism Studies*, 5(1), 87–100.

Karlsson, M. (2011) "The immediacy of online news, the visibility of journalistic processes and a restructuring of journalistic authority." *Journalism*, 12(3), 279–295.

Karlsson, M. (2012a) "Charting the liquidity of online news: Moving towards a method for content analysis of online news." *International Communication Gazette*, 74(4), 385–402.

Karlsson, M. (2012b) "The online news cycle and the continuous alteration of crisis frames: A Swedish case study on how the immediacy of online news affected the framing of the swine flu epidemic." *Journal of Organisational Transformation & Social Change*, 9(3), 247–259.

Karlsson, M. and Sjøvaag, H. (2016) "Content analysis and online news." *Digital Journalism*, 4(1), 177–192.
Karlsson, M. and Strömbäck, J. (2010) "Freezing the flow of online news: Exploring approaches to the study of the liquidity of online news." *Journalism Studies*, 11(1), 2–19.
Karpf, D. (2012) "Social science research methods in internet time." *Information, Communication and Society*, 15(5), 639–661.
Lewis, S. C., Zamith, R. and Hermida, A. (2013) "Content analysis in an era of big data: A hybrid approach to computational and manual methods." *Journal of Broadcasting & Electronic Media*, 57(1), 34–52.
Lim, J. (2012) "The mythological status of the immediacy of the most important online news." *Journalism Studies*, 13(1), 71–89.
Martin, F. (2015) "Getting my two cents worth in: Access, interaction, participation and social inclusion in online news commenting." *#ISOJ*, 5(1), 80–105.
Sjøvaag, H. and Stavelin, E. (2012) "Web media and the quantitative content analysis: Methodological challenges in measuring online news content." *Convergence: The International Journal of Research into New Media Technologies*, 18(2), 215–229.
Sjøvaag, H., Stavelin, E. and Moe, H. (2016) "Continuity and change in public service news online." *Journalism Studies*, 17(8), 952–970.
Steuer, J. (1992) "Defining virtual reality: Dimensions determining telepresence." *Journal of Communication*, 42(4), 73–93.
Tumber, H. (2001) "Democracy in the information age: The role of the Fourth Estate in cyberspace." *Information, Communication and Society*, 4(1), 95–112.
Widholm, A. (2016) "Tracing online news in motion." *Digital Journalism*, 4(1), 24–40.
Zamith, R. (2016) "On metrics-driven homepages." *Journalism Studies*, 1–22 [advance online publication].
Zamith, R. (2017a) "A computational approach for examining the comparability of 'most-viewed lists' on online news sites." *Journalism & Mass Communication Quarterly*, 1–20 [advance online publication].
Zamith, R. (2017b) "Capturing and analyzing liquid content." *Journalism Studies*, 18(12), 1489–1504.
Zamith, R. and Lewis, S. C. (2015) "Content analysis and the algorithmic coder: What computational social science means for traditional modes of media analysis." *The ANNALS of the American Academy of Political and Social Science*, 659(1), 307–318.