2011 11th IEEE International Conference on Computer and Information Technology

Survivable Interaction Distribution Networks

Shun-Yun Hu
Institute of Information Science
Academia Sinica
Taiwan, R.O.C.
Email: syhu@iis.sinica.edu.tw

***Abstract*—Interaction Distribution Networks (IDNs) are systems designed to deliver small, interactive data streams in a scalable manner. Compared with Content Distribution Network (CDN), which serves relatively static and bulky data, IDNs may be more suited for latency-sensitive bi-directional interactive streams such as network games or real-time conferencing.**

Although existing instant messenger (IM) or massively multiplayer online games (MMOGs) may be considered as early forms of IDNs, we envision IDNs as capable to serve new forms of scalable interactions to different applications at affordable costs. Designing IDNs that are both scalable and reliable thus is important. This paper discusses the survivability of IDNs and the design measures that can be taken, to ensure that an IDN may still survive hardware and software malfunctions, when deployed on a large scale.

***Keywords*-publish/subscribe, virtual environment, instant messenger, interaction, survivability**

I. Introduction

The Internet was designed as a scalable system suitable for various data flow. Since the creation of World Wide Web (WWW), *information* in the forms of web pages, pictures, videos, etc., has become much more easier to access, share, and store. However, besides information, another major function of the Internet—-real-time *interactions*—-still has much to explore.

Currently there are deployed systems that support real-time interactions for millions of concurrent users. Most notably are the *instant messenger* (IM) [1] networks such as Yahoo Messenger, AIM, MSN, and Skype. However, most of the interactions occurred on IM networks are within a small group (i.e., between two to a few tens). Another type of large-scale interactions occurs in Massively Multiplayer Online Games (MMOGs) [2] such as World of Warcraft or social virtual worlds such as Second Life [3]. However, while the total concurrent users can be impressive (e.g., World of Warcraft has reported over a million concurrent users [4], and Second Life has around 90,000 peak concurrent users in the entire virtual world [5]), the scale of interactions within these systems are typically within 100 users for a given region / scene / interaction group.

It is therefore not yet possible, to hold live events on a large scale on the Internet. For example, a popular lecture attended by thousands and anyone from the audience can ask live questions; a rock concert with tens of thousands of participants who can engage and *feel* the atmosphere of being with the star; or a gathering of emergency rescuers immediately after a natural disaster, for briefing and discussions on how to proceed with the rescue. These *large-scale interactions* currently can only happen in the physical world, when a large enough physical space is provided, and all participants of the event have gathered at the same place. The requirement of physical presence, has more or less limited the number of potential participants and the occurrences of such events. On the other hand, if a general mechanism to support large-scale interactions were available on the Internet, we may see fundamental and widespread changes in how people interact, from music concerts to political rallies.

We postulate that there are some fundamental similarities between IM and MMOG systems, and systems that support real-time interactions in general. Once identified, we can design and extend such *interaction distribution networks* (IDNs) [6] to support real-time interactions in a more scalable and flexible manner. Similar to existing *content distribution networks* (CDNs) [7], which serve content files such as web pages, audio and video streams on massive scales, once IDNs can be constructed, they may also serve interactions in a scalable manner. Unlike the uni-directional data served by CDNs, however, the data served by IDNs should be bi-directional interactive streams that come in short bursts of small packets (instead of the relatively static, and sometimes bulky content CDNs are designed to support).

In order for IDNs to be universally suitable for different applications needs, the key requirements of IDNs are interactivity, scalability, and generality [6]. However, for any system designed to operate on a massive scale, it is also important to consider its reliability, or *survivability*, in face of hardware and software failures. In this paper, we propose that to fulfill this goal, it is important to consider three main issues: functional redundancy, attack tolerance, and inter-operability. We can borrow concepts from *peer-to-peer* systems [8], where each unit has similar functionalities (e.g., can be both a client and a server at the same time), and a single point of failure is generally avoided.

In the rest of this paper, we first provide background on IDN in Section II, and explain some IDN basics in Section III. We describe key requirements for IDN survivability in Section IV, and conclude this paper in Section V.

978-0-7695-4388-8/11 $26.00 © 2011 IEEE
DOI 10.1109/CIT.2011.81

II. Background

A. CDNs

Content Distribution Networks (CDNs) [9] are systems designed to facilitate the delivery of web content. Soon after the introduction of WWW, people discovered that web servers may be overloaded and cease to function when high popularity content existed (i.e., when *flash crowd* occurs for a site). CDNs then were introduced to alleviate this problem by redirecting original requests to one of the *edge servers* closest to the requesting user. Content is replicated at the edge servers and also synchronized whenever new content becomes available. However, while CDNs account for a fair amount of web traffic today, the data served is still mostly static content in a uni-directional manner. Interactions, which are bi-directional and may come in short bursts of packets, are not primarily supported by CDNs.

B. IM and MMOG

The most deployed and largest interaction networks today on the Internet may be the Instant Messenger (IM) networks, where concurrent users can be in the range of tens of millions for a given system [1]. Users generally need to obtain a service-specific account, to login the system and build their networks of friends and associates. Direct one-to-one communication via text or voice are the most common form of interactions, while small groups of tens of people may also jointly participate in a group discussion. IMs may find their precursors in Internet Relay Chat (IRC) channels, which are operated via a network of volunteer service providers. However, while successfully adopted, today's IM networks still support only small interaction groups (i.e., group size of a few tens, and less than 100 in general). Architecturally, each interaction group is hosted on a specific server, so the capacity of a given server may limit the number of potential participants. The more bandwidth-intensive the interaction, the smaller the group size (e.g., an IRC chatroom can host over a hundred of users, while a Skype channel typically can host only ten people). IM networks also do not support interactions beyond simple message transmissions.

Massively Multiplayer Online Games (MMOGs) [2] on the other hand, support more diverse forms of game-specific interactions that include fighting, trading, building, or using items, etc. Such additional capability is made possible by having servers that execute specific game rules (or *game logic*) given certain *event* messages and the current state of the game, as stored in game objects. While architecturally more flexible, the workload of an MMOG is often divided by spatial partitioning of the virtual world, where each *region* is assigned to a particular server to manage. Server overload or underload (i.e., idling) thus can be common, given the diverse distribution and density of users within the world [10]. Group interactions are still limited to less than 100 users in general, to avoid CPU or bandwidth over-use.

C. Interest Management

One key to scale up MMOG systems is based on the concept of *interest management* [11]. While the total number of concurrent users in a system may be large, it is recognized that the scale of direct interactions for each user is often small, so filtering can be done at the server to deliver only messages of interest to a user. For example, if a user is only interested to see or interact with other users within a 50 meter radius, such *area of interest* (AOI) can provide the server a filtering criteria, to decide which state updates are relevant and should be sent to the user.

While users may avoid data overload in this manner, the filtering tasks itself demand computation and constitute a potential overload for the server. Najaran and Krasic thus have suggested that by separating out the filtering (i.e., interest management) from the processing of game logic (or *state management*) [12], larger interaction group can be achieved. Lake et al. [13] also suggest that by separating the server component handling client connections from the main server functions (i.e., state and interest management), scalability of the system can further be improved.

D. Publish / Subscribe

One common way to realize interest management is via the publish / subscribe (pub/sub) paradigm [14]. The simplest form is *channel-based* (or topic-based) pub/sub. A *subscriber* would specify that she is interested in the messages of a particular channel by expressing a subscription interest. Then whenever new messages are delivered (i.e., published) to that channel, all the subscribers of the channel would receive the message.

In the case of IM, each interaction group can be seen as a message channel, where all participants can subscribe and publish messages to. For MMOGs, a common technique is to assign each partitioned region its own channel, and have the server listen to all event messages delivered to that channel. User clients can also subscribe to channels that constitute its AOI, to receive relevant update messages. However, in such case, the basic problem is how to find the right partitioning, so that overload or underload of the server (i.e., the region's channel) will not occur. Spatial publish / subscribe (SPS) [15] is another form of pub/sub where the filtering is not based on specific channels, but on spatial areas. Whenever a publication area overlaps with a subscription area, the published message is delivered to the subscriber. A user can perform an *area subscription* to indicate an interest in all update messages within a certain area, while publish his actions as event messages as *point publications*. While the server can perform *area subscription* for event messages generated by users, and publish state updates as *point publications*, to be received by user clients. Viewed in this way, the scalability of a given interaction group thus depends on how scalable the pub/sub service can be.

III. IDN BASICS

We define an IDN as a system that supports real-time interactions between human participants on a large-scale [6]. The *scale* here refers to both the total number of system participants (i.e., *system scalability*), and the number of participants within an interaction group (i.e., *group scalability*). For example, a good IDN should be able to serve, both thousands of participants in a live concert, and millions of concurrent users across the entire system.

By *interaction*, we mean small data packets that represent human behaviors (e.g., text and voice speech, gesture data, or more sematic data such as trading and attacking in games). Often the interaction is bi-directional and within a certain *interaction group*. We observe that most interactions can be described by an *event* $\rightarrow$ *processing* $\rightarrow$ *update* data path. For example, in a MMOG, when user A trades a sword with user B for money, a *trade event* is first sent from the user to the server, where the event is checked for correctness (e.g., whether both users have the items, and whether a trade is allowed given the game logic). If the event is accepted and properly processed, some game states will be changed, and *updates* are sent to the affected users (e.g., user A is notified of some money increase in her pocket, while user B receives a sword). For simpler interactions, such as a chat on IM networks, the *processing* stage may be absent and a chat message (i.e., sending of an event) can go directly to the affected user (i.e., receiving of an update). In its most general form, the *event* $\rightarrow$ *processing* $\rightarrow$ *update* data path may be observed for all interactions. IDNs thus should be designed to support this data path, while meeting its basic requirements.

A. Requirements

Interactivity As interactions are exchanges between human users, it is important that they perceive the process as happening in real-time. Interactions thus are latency-sensitive and should complete within some well-specified time limits. However, the exact latency tolerable depends on the interaction or even application type. For example, voice packets should arrive within 100ms, while a latency up to a couple of seconds may be acceptable for text chat.

Scalability The scalability requirement for IDNs dictate that both *system scalability* and *group scalability* should be achieved. This requirement means that an interaction data path should still be completed within a given latency limit, even when the number of users increases within the group, or within the system. In terms of system design, this means that when the number of user scales, the system resources devoted to support these users should also increase accordingly. Ideally, resource requirement and provisioning should match with each other (e.g., both can scale linearly, as opposed to a quadratic increase in consumption is met with only a linear increase in provisioning).

Generality Although existing IM and MMOG systems do fulfill some interactivity and scalability requirements as mentioned above, often they are specialized systems designed for a particular purpose. We do not yet see interactions that are supported in a generic manner, much like how CDNs have helped the Web to distribute content. A system that can serve different applications *on the same infrastructure* thus would be hallmark for an IDN.

B. IDN Components

To realize an IDN, we may generalize its components and potential design from existing systems. The key to interactivity is to ensure that both data delivery and processing can be completed within bounded amount of time, which requires that these tasks be distributed to different nodes, when the number of user scales. As long as the number of users within an interaction group is small, system scalability can be achieved by simply providing more server machines for each interaction group. However, the assignment of workload to servers should be dynamic and flexible, so that we do not face server overload or underload. Group scalability on the other hand is more tricky, and some kind of forwarding mechanism may be needed (e.g., [16]), so that users may still receive relevant messages within the same group, at the expense of slightly increased latencies.

The central design idea for an IDN is that the processing of states (e.g., *state management*) is separated from the expression of interests (e.g., *interest management*), so that overall scalability may increase, without hindering or overloading any particular system component. We identify three main components that an IDN can include and describe them as follows.

Proxies Proxies are servers where the user clients connect to directly, in order to join the IDN. Users should connect to the closest proxies (latency-wise) in order to minimize overall latency. IDN proxies however differ from CDN proxies in that instead of serving content, they perform publish / subscribe requests on behalf of the users, to other parts of the IDN networks (namely the matchers).

Matchers Matchers are server that support the publish / subscribe functions, so proxies (on behalf of the users) may specify *interest expression* in the form of subscription requests to the matchers. User events are also sent (i.e., publish) to these matchers via the respective proxies, so that the event messages can be re-directed to appropriate handlers (i.e., subscribers) for processing.

Managers Managers basically serve the function of object state maintenance (e.g., MMOG game objects), and the processing of events to modify the object states. Once states are modified, *update* messages may then be published by managers to the matchers, so that interested users can receive the updates via their proxies' subscriptions.

C. Design Considerations

To ensure that interactivity is achieved, the time taken for an *event* → *processing* → *update* cycle should be bounded within some limit, this implies that the time spent for interest management and state management, should both be within certain limits. The processing time taken by state management is a function of the number of data objects accessed and updated, and will depend on the nature of the interaction and processing logic (i.e., event handling rules). Assuming that only a limited number of objects are accessed for any given event, the time for state management may be relatively constant. On the other hand, the time taken for interest management (done at the matchers) is a function of the interaction group size, so the larger the group, the longer the delivery time.

One way to manage delivery delay, as group size grows, is to make matchers deliver messages not directly to subscribers, but some helpers first, which then help to forward the message (e.g., a *forwarding pool* formed by machines with good upload capacities [17]). The amount of message deliveries thus may be reduced, at the expense of increased latencies. The additional latencies tolerable and the number of helpers that can be found, will determine the limit of the interaction group supportable.

IV. Survivability Requirements

We define *survivability* as the ability of the system to sustain a certain amount of component failures, yet still be functional. For a well-designed system, it is only possible to break it if a large ratio of its components (nodes) have failed. The design of the Internet may be the best example of a survivable network, and the key feature is the avoidance of any centralized component throughout the system. The Internet is composed of routers which all speak and share the Internet Protocol, and its main function (i.e., data delivery) can be performed without any central resource. So single or even regional failures of routers do not hamper the system, as data delivery can be supported by still-functional routers. Extending from this concept of single point avoidance, we consider that there are three important design considerations for the survivability of an IDN: functional redundancy, attack tolerance, and inter-operability.

A. Functional Redundancy

Existing systems that support interactions often have a single point of entrance, for example, a game lobby server in the case of MMOG, or an initial contact server in case for IM services. Such an entry point thus constitutes as a potential bottleneck for the system. When faced with a large number of concurrent join or denial of service attack, the system may temporarily cease to accept new join requests and can be considered non-functional by its users. One solution is to make bootstrapping (i.e., joining the system) done via multiple, equally valid entry points. Having a list of available proxies in this case would help. However, other issues such as unique ID assignment and entry point discovery, should also be distributed and do not rely on centralized resources.

Once users have login into the system, the *event* → *processing* → *update* data path of an IDN is executed over three components: proxies, matchers, and managers. An IDN thus can be most fault-tolerant, when each of these components is replaceable by other nodes of the same type. To do so, redundant functional components that can replace one another's function should be built into the system.

Ideally, *each* execution of an *event* → *processing* → *update* cycle should be capable of going through a different set of system components (e.g., proxy, matcher, and manager). This is also the rationale behind new MMOG architectures such as Sun Microsystem's Project DarkStar [18]. However, as there are costs in discovering and selecting the nodes to use, normally the system operates most efficiently if the same set of nodes can be repetitively used, avoiding discovery and selection costs. Replacement should only occur when the data path is blocked and a failing component needs substitution by a functioning alternative.

Failure recovery deals with two main issues: whether the states at the component is recoverable, and whether other interacting nodes may detect the failure and contact the replacement. The less stateful a component, the easier its recovery or substitution when failure occurs. We discuss the possible failure scenarios and recovery mechanisms for each:

Proxy Proxies are stateful of the subscription records of the user clients they serve. However, as such records are also available at the users, if a proxy fails, the connected user client should simply find another proxy and perform a re-subscription. This in practice can be considered as a system re-join by the clients. While some costs exist, no extra functions are needed for proxy recovery. When a substitute proxy is up, the new proxy can perform re-subscription on behalf of its user clients to the matchers, so that the subscription records at the matchers are also updated.

Matcher If a matcher fails, the data lost is mainly the pub/sub information stored at the failed node. Specifically, subscription records are more important than publication records, as publications are designed to be short-lived and non-stateful. If a given publication is lost, the effect can be duplicated with a simple message re-publication. In theory, if the subscribers or publishers can detect the failure of the matchers, then they can perform re-subscription or re-publication of the messages. So even though the matchers are stateful (by having subscription records), such records can be re-constructed as duplicated information exists at the original subscribers or publishers (i.e., other proxies or managers). The key issue then is that proxies and managers need to detect the failures of the matcher(s) in contact, and re-perform publications or subscriptions in a timely manner.

Manager Managers are the most stateful among the three main IDN components, as they need to maintain object

states meaningful to the sematics of interactions. There is thus no straightforward recovery for their failures and certain fault-tolerant mechanisms need to be designed in place. The most basic measure is to duplicate the states stored at each manager at their neighbors. The *neighbor* here can simply be another well-defined manager. For example, for channel-based pub/sub records, the neighbor can be the *successor* node on a DHT ring [8]; for spatial pub/sub, the neighbors can simply be the managers responsible for the enclosing areas. When a given manager fails, its neighbors thus can detect and takeover its original tasks, and also perform re-subscription of its interests to the matchers. How fast or thorough the recovery can be, will depend on how complicated the object states are being stored. The simpler and less stateful the objects, the easier the recovery (e.g., in an MMOG, objects that store only user positions are easier to recover than objects that store the status, inventory, equipments of users). As a secondary measure, the objects at the managers can be backup to persistent databases periodically, so that in case of disaster or massive failures (e.g., failures of a manager and its designated backup neighbors at the same time), the object states are still eventually recoverable from the database. Reliable and fault tolerant database then becomes a requirement for system reliability. But here we assume that such database exists and can be integrated to the IDN.

B. Attack Tolerance

Large-scale services today face many forms of attacks, we discuss how IDNs may survive malicious attacks (e.g., denial of service, or DoS attacks) launched possibly from multiple sites. Functional redundancy already provides a certain protection against attacks. However, if certain proxies are targeted by a DoS attack, besides having the clients try to join the system via different proxies, more active approaches may be needed. A technique used by existing websites for load balancing and fault tolerance is to provide multiple service points, and have the DNS to return different IP addresses for the same hostname, so that user traffics may be re-directed to different service points (also called the Round-robin DNS). However, as DNS records can be cached by clients or within the DNS hierarchy, it is possible that clients will still attempt to connect to obsolete / failed service points.

A more active approach is to segment the joining process into stages, where in the first stage, the client first contacts one of the known proxies, but only to obtain information of another live proxy, with which subsequent connections will occur. This way, only authenticated (legal) clients may learn of the proxies that perform the actual services, while publicly known proxies only serve the pointers. This way, not only the publicly known proxies can have lighter loads, if they are being targeted for DoS attacks, they can also shutdown and change IP quickly and easily.

Assuming that there is a large IP pool at the disposal of the IDN service provider, where at any given time only a subset of the IPs are alive, and that clients refresh their known list of public proxies every time they connect to the system. The system can increase the variety and flexibility of its service points, and make it harder to have a targeted attack at the system.

C. Inter-operability

The above two considerations, if fully implemented, already provides a certain degree of survivability against failures. However, while these measures deal with the failure within a single IDN, survivability at a higher level demands that different IDN systems can share common vocabularies (i.e., message formats and protocol) such that the entire IDN services can still be operational even if individual IDNs have failed (i.e., inter-operability exists among them).

The concept here is very much akin to how the IP-based Internet itself is resilient against failures of any number of its components (i.e., routers), and how WWW is resilient against the failures of any number of websites. While individual websites are failable, the entire WWW is difficult to fail as many different websites exist, and broken links do not affect the main functions or usability of the whole Web system. This brings us to the interesting issue that whether IDNs can be constructed similar to WWW in that common functionality exists between different IDN services and users can simply migrate from one to another easily. To the users, this effectively means that there is just one global IM service and users of different IM systems can all talk to each other; or in the case of MMOG, users can actually migrate between different MMOG systems, going from one system to the next, with ease and possibly using the same virtual avatar.

We leave this as thoughts for the future, as this is not just a technical issue, but involves non-technical considerations (e.g., IM operators generally want to segment their users and have non-interoperable systems, so that they can attain and keep user records to themselves; similarly, a *universal game client* may not be desirable to MMOG operators, if users who can easily leave are more than users who can easily enter). However, as we have seen examples of such system (e.g., WWW and the Internet itself), that by networking all systems together, the total utility is greater than the sum of its parts, such prospects are not entirely impossible, if operators can find out that there is more to gain than lose by being inter-operable with one another.

To the users, being interoperable at the IDN level means that he or she can use a service and migrate to the next seamlessly. Chatting or finding IM friends at another network, or going from one MMOG to another, is as seamless as going from one website to the next.

V. Conclusion

While we do not yet see general Interaction Distribution Networks (IDNs) today, their emergence may be expected as demands for real-time Internet interactions increases. In this paper, we have briefly described the components of an IDN and how it can provide scalable interactions to both existing IM and MMOG systems, while enabling larger group sizes. We also identify that the survivability of an IDN will mainly rest on three issues: functional redundancy, attack tolerance, and inter-operability. Among them, the redundancy of managers may be the most complex, as it involves how application-specific object states should be replicated and recovered; realizing inter-operability is also non-trivial, as it is not just a technical issue, but social and political as well.

Existing CDNs are operated as commercial services that facilitate the distribution of web content, the operation of IDNs may work in a similar fashion. However, for IDNs to be truly scalable and survivable, inter-operability among different IDN components may be needed, so that users can easily migrate and interact with users from different IDNs, and form interaction groups dynamically. Failures of IDN components may also be most recoverable, given the large pool of units with similar functions. In time, the emergence of IDNs may enable new scalable applications and behaviors on the Internet that we have yet to experience.

References

[1] IM, http://en.wikipedia.org/wiki/Instant_messaging, 2008.

[2] T. Alexander, *Massively Multiplayer Game Development*. Charles River Media, 2003.

[3] P. Rosedale and C. Ondrejka, "Enabling player-created online worlds with grid computing and streaming," Gamasutra Resource Guide, 2003.

[4] http://wow.joystiq.com/2008/04/11/chinese-wow-hits-1-million-concurrent-players/, 2008.

[5] http://alphavilleherald.com/2009/12/second-life-losing-traction-concurrent-users-slide.html, 2009.

[6] S.-Y. Hu, "Interaction distribution network," in *Proc. International Conference on Digital Information Processing and Communications (ICDIPC 2011)*, 2011.

[7] L. Wang, K. Park, R. Pang, V. Pai, and L. Peterson, "Reliability and security in the codeen content distribution network," in *Proceedings USENIX Annual Technical Conference (ATEC '04)*, 2004.

[8] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," in *Proc. SIGCOMM*, 2001.

[9] http://en.wikipedia.org/wiki/Content_delivery_network, 2011.

[10] Y.-T. Lee and K.-T. Chen, "Is server consolidation beneficial to mmorpg?" in *Proc. IEEE Cloud*, 2010.

[11] K. L. Morse, L. Bic, and M. Dillencourt, "Interest management in large-scale virtual environments," *Presence*, vol. 9, no. 1, pp. 52–68, 2000.

[12] M. T. Najaran and C. Krasic, "Scaling online games with adaptive interest management in the cloud," in *Proc. NetGames 2010*, 2010.

[13] D. Lake, M. Bowman, and H. Liu, "Distributed scene graph to enable thousands of interacting users in a virtual environment," in *Proc. NetGames 2010 (MMVE session)*, 2010.

[14] A. R. Bharambe, S. Rao, and S. Seshan, "Mercury: A scalable publish-subscribe system for internet games," in *Proc. NetGames*, 2002, pp. 3–9.

[15] S.-Y. Hu *et al.*, "A spatial publish subscribe overlay for massively multiuser virtual environments," in *Proc. 2010 International Conference on Electronics and Information Engineering (ICEIE 2010)*, 2010, pp. V2–314 – V2–318.

[16] J.-R. Jiang, Y.-L. Huang, and S.-Y. Hu, "Scalable aoi-cast for peer-to-peer networked virtual environments," in *Proc. ICDCS Workshop Cooperative Distributed Systems (CDS)*, 2008, pp. pp.447–452.

[17] A. Bharambe, J. R. Douceur, J. R. Lorch, T. Moscibroda, J. Pang, S. Seshan, and X. Zhuang, "Donnybrook: Enabling large-scale, high-speed, peer-to-peer games," in *Proc. SIGCOMM*, 2008.

[18] J. Waldo, "Scaling in games & virtual worlds," *ACM Queue*, Nov/Dec 2008.